TALLINN UNIVERSITY

School of Digital Technologies

Digital Learning Games

Methodyca: A Study of the Development Process of a Game about Research Methods

Master's Degree Thesis

Authored by: **Ahmed Mohamed Said Anwar ElShenawy**

Supervised by: **Martin Sillaots, PhD**

Tallinn, 2021

ACKNOWLEDGMENTS

I would like to thank the Methodyca team for all the hard work they put in making it what it is today. Methodyca as a game and research project was made possible by the sponsorship of HARNO IT Akadeemia (project number 7-1.1/1-20).

I would also like to express my gratitude for my supervisor Martin Sillaots, PhD. His contributions to the making of Methodyca and this thesis have helped in many ways to realizing it.

Lastly, I would like to thank all my friends who helped along the way, and my dearest thanks to my partner, who has only shown love and patience throughout this entire journey.

CONTENTS

LIST OF FIGURES

ABSTRACT

This thesis aims to contribute to the body of research and academia related to the development processes of digital games, particularly through the analysis of the development of the main systems comprising Methodyca; a point-and-click adventure game which mediates learning about research methods. This study adopts the Design through Research methodology, and represents the 4th step in that methodology: Reflect and Disseminate. Methodyca's systems are examined through a retrospective review of how they were created, and the choices and decisions that went into making them, by the development team, while looking at how play-testers perceived and played the game, then linking their feedback to its respective game systems. The systems created and adopted for Methodyca are most notably flexible and modular, which allow developers and designers to either expand Methodyca itself, or use the same foundations to create similar point-and-click adventure games. This thesis, along with Methodyca as a game and its publicly available project files, all come together as a whole to allow the reader to understand the development process that went into this game, in addition to providing them with the knowledge to recreate the systems in other contexts, as well as improve on them in future iterations. Thus, concluding the Research through Design strategy applied in this research through its last step: to *Repeat* the knowledge and take it even further.

INTRODUCTION

Game development projects in university courses, such as Tallinn University, tend to be small, simple, and focusing on one main aspect or design element. Examples can be found at the Digital Learning Games' (DLG) portfolio page. One reason behind this phenomenon is that these projects tend to span from 1 week to 3 months, depending on the assignment or course requirements. This diminishes the chances for students to learn and experiment with developing multiple complex systems in a game project; such as state saving system, dialogue and character interaction system, or implementation of advanced game mechanics. Furthermore, most research literature produced about digital games and game-based learning is mainly concerned with the design aspects of the games, the effectiveness of their design strategies, or the outcomes of playing the games based on their design. There is very little literature that actually focuses on the game development process and practice, with only 48 research papers out of over 2,000 publications talking directly about game development between the years 2006 and 2016 (Berg Marklund et al., 2019).

Game design is without doubt an important and critical topic; it determines what the user sees and hears, how the game is presented, how it is controlled, how the user is expected to interact with it, what are the actions and reactions a game should do, and what are the outcomes of experiencing a game. The design is the tangible aspect of the game, which players are able to discuss with each other and provide feedback to the designers about. However, behind each and every aspect of these game design elements is a piece of computer code that drives it. Without the hundreds of lines of code written by the game programmers, all that complex design will either end up making a tabletop board or card game, or simply remain as a well-documented paper prototype – there can be no digital game. Therefore, studying how digital games are *made* is as necessary as learning how to design them.

When considering other studies in the topic of game development; these examined planning, management, team work, and other elements such as the work environment and stakeholder management (Borg et al., 2020; Kortmann & Harteveld, 2009; Osborne O'Hagan et al., 2014).

These kinds of studies are useful in analysing the situations, environments, and conditions that games are typically created within, but they do not look at the product itself. In their paper, Berg Marklund et al. (2019) mention a notable problem with current research: most studies are performed by academic researchers that rarely take professional game makers' consultation into account; thereby missing on or ignoring a lot of the nuances of game development which occur in practical conditions.

Further studies and research should address the process of designing the systems that drive the games themselves, independent of the conditions and environment they were created under. It is necessary to explore how different game systems are made and the technologies driving them, such as how game engines affect the game development process (Freedman, 2018) and third-party tools or plugins. These kinds of studies can help students of game development to look at previous designs, build upon, and improve them, instead of attempting to reinvent the wheel, or worse: repeat the same mistakes.

One potential reason that makes studying game development processes and techniques a laborious undertaking is because there are no established methods or common language among developers when it comes to game development (Berg Marklund et al., 2019). Furthermore, academics conducting game-related studies don't necessarily know or understand how to develop a digital game (P. Martin, 2018), in order to be able to examine and study one. In the same manner, access to the source code or project files of already (or to be) published games by professional game studios, in order to analyze them, is a largely complicated matter, as these are considered to be trade secrets and competitive advantages, further encumbering the efforts to discuss the making of games. Some game makers do offer their code for others to check and learn from, like the ones who participate in game development hackathons called game jams – the Global Game Jam and Ludum Dare are two examples. However, these games tend to be overly simple and usually created very quickly under time pressure (Borg et al., 2020), thus not providing feasible content for proper game development studies and research. These factors add up to making research into the details of game building and development all the more troublesome.

To this end, this thesis contributes to the research field by examining and analyzing the development process and results of one digital game named "Methodyca"; a web-based digital learning game conceived, designed, and developed by a team of Master students and professors at Tallinn University. What sets Methodyca apart from other game projects made during an academic study period is that:

- Methodyca was created over the course of more than a calendar year, making it a far bigger and more ambitious project than those typically created in the span of weeks or a couple of months;
- Methodyca was designed not as a course assignment, but as a game-based learning product to be actively used by students (and enthusiasts) to help them learn about research methods; and
- Methodyca is an amalgamation of several game systems and mechanics, that looks and plays very similarly to commercially-released games, instead of focusing one a single game design element or mechanic.

This makes it a worthwhile game to study the development process of, given the added benefit of myself being the lead developer of the game, with full knowledge of all the major systems that make it work and sets Methodyca apart as mentioned above. Furthermore, this study hence addresses one of the issues stipulated by Martin (2018) and Berg Marklund et al. (2019), where most of the available research in game studies is carried out by academics with little knowledge about the process of game making. It also addresses the other problem mentioned above, where a game's source code and programming methods aren't available to the authors of that game's study, due to trade secrecy and non-disclosure agreements (NDAs) signed by the developers; because Methodyca's source code is made public.

The main purpose of this thesis is to provide insights and discussion in the development process of a medium-sized digital game. These insights can be helpful to successive students working on game projects of any size; to learn the concepts behind building game systems similar to those in Methodyca, to improve these systems and address either problems or opportunities in them, to know what game-making engines and tools would be more suited to their project's needs,

and to avoid making the mistakes that have been (or could have been) made during the development process of Methodyca.

Hence, the goals of this study are:

1. A comprehensive examination of the overarching systems and concepts used in the development of Methodyca.
2. Provide game developers and readers who are beyond the beginner level (in game development) with ideas and tips for game development – specifically point-and-click adventure games, which is the way Methodyca is designed to be played as.
3. Determine the shortcomings in Methodyca's systems, and propose other alternative approaches and workarounds.

Much like how a game designer wonders about how to design a game's playable system, a game developer also has to understand how to implement that system via computer code and the tools they're using. In that regard, the study of Methodyca seeks to answer the following questions:

1. What are the common elements/systems found in point-and-click adventure games?
2. How to build a modular and expandable point-and-click adventure game?
3. What are the lessons learned about the process of game development from player feedback and this retrospective study?

As stated before, Methodyca is already developed and available to play on its official website, likewise its source code is made available online via the GitHub source control service (link here) for anybody to download and delve into. This study will be mainly concerned with examining the major systems that were created in the game, such as the inventory and state saving systems. In addition to the self-reflective exploration of Methodyca, play-testing feedback was collected via questionnaires and live sessions, as well as observing how the participants interacted with the game, in order to assess the performance of these systems when played by potential end users.

1. METHODOLOGY

This chapter describes how the research study was carried out, what is to be expected in this study and what this study does not include. It gives a brief overview about Methodyca, the digital game that is the object of this study, and what systems will be examined in it, as well as the process by which that examination will take place – this dictates the subsequent chapters of this document.

It was planned to have participants play-test Methodyca in person with the presence of the developers and designers of the game. However, due to the COVID-19 pandemic's restrictions, the testing model was changed into one where participants were asked to do the testing individually at home and using their personal computers, while answering an online questionnaire as they play-tested Methodyca. Nevertheless, some live testing sessions in person were possible to be carried out, but with a smaller number of participants than originally planned.

Finally, this chapter illustrates how the game data and participants' feedback were collected and analyzed, and how the test version of Methodyca was modified from the official one available to the public, in order to facilitate the testing of the game's systems and functionality and skip the parts irrelevant to the purpose of this study.

This study will be mainly concerned with examining the major systems that are included and functional in the game, such as the inventory and state saving systems. This study will not address elements such as setting up game objects; as these are not only minor aspects relative to the other more complex systems in Methodyca, but are also features that can be easily found in many online tutorials. Similarly, functionality that was either dropped or didn't make it to the final released product will not be analysed in this document.

One important clarification to make is that the systems will be explored and described in a somewhat high-level manner; meaning that special care will be taken to explicitly *avoid* discussing the programming language and the game engine or tools used to build the systems,

unless it is felt necessary to do so. The reason behind this choice is to make these systems agnostic of programming languages and specific tools, hence preventing the reader from falling into the trap of assuming that there is only a single approach or tool to achieve these results, or from becoming distracted with secondary issues such as coding rules and syntax – where programming techniques are not the goals of this study. In any case, if the reader still wishes, and they are in fact invited, to go through the game's code and tools used, then they can freely access it at the GitHub online repository (https://github.com/AShenawy/RM-Game). The source code is made available to enable anybody to read the code, copy it, or improve it (subject to its licensing agreement).

1.1. Strategy

The method used in this study is Research through Design (RtD), as it was termed by Christopher Frayling (1993). As Zimmerman et al. (2010) put it: RtD is a 'research approach that employs methods and processes from design practice as a legitimate method of inquiry'. In RtD, an artefact is created to answer a question or a problem, but the focus is on the knowledge gained and imparted *through* the process of designing and building iterations of the artefact until the final product is reached. Borrowing from the 5-step setup by Zimmerman & Forlizzi (2014) in carrying out RtD research project, this study constitutes the 4th step – 'Reflect and Disseminate'. The aim is to reflect on the work done so far, and the experience gained from it. The previous 3 steps are present in how the systems building Methodyca were selected (1st step), designed (2nd step), and evaluated (3rd step) to confirm they were working as intended for the requirements they were set out to achieve in making the game function as required. Indeed, how Methodyca is at its current state is not the work of a single measure-and-cut attempt, but the culmination of several repetitions of design, build, and test loops. The final step in the RtD project is to repeat. This step is intended to be carried out by fellow scholars and developers, who may take this work and re-apply it in new situations and conditions, to then add their input and improve the overall design of the systems developed here. And even Methodyca, while now released, doesn't represent the ultimate version of itself, as it would also go through new cycles of revisions and improvements.

RtD should not be confused with Research *by* Design, which happens to closely resemble RtD and yet is not exactly the same. Research by Design is meant to define the connections between design and research, and to explore the 'different materials by which a design is carried out' (Roggema, 2016). RtD also varies from research *into* design and research *for* design, which are concerned about the research process and the final product, respectively (Frayling, 1993).

1.2. Study Object

Methodyca was a project conceived by the Digital Learning Games department of the School of Digital Technologies in Tallinn University. The aim was to create a game-based learning experience, to assist higher education students with their studies of research methods.

Figure 1: Screenshot of a Methodyca game screen (artwork by Anni Hõ)

Methodyca is a point-and-click adventure game, where the goal of the game is to find the player's missing supervisor who presumably travelled through a portal to an unknown universe. In the course of their search, the player encounters different characters which represent real-world scholars, such as Margaret Mead, an American cultural anthropologist, and William Sealy Gosset, an English statistician, chemist, and head brewer of Guinness who pioneered experimental design and analysis. In order for the player to progress through the game, they

have to play and win mini-games that represent different research and data collection methods, including questionnaires, interviews, and prototyping. In the process, players are expected to apply their knowledge from the research methods university course, as well as experiment with it in the game. Some minigames are relevant to both quantitative and qualitative research methods (like prototyping), while others belong to either quantitative only (survey) or qualitative only (interview). In that regard, one unique twist in Methodyca is handing the player a special device that teleports them between two parallel dimensions, quantitative and qualitative, to play their associated minigames; where despite these two dimensions looking visually similar, they still have different characters and mini-games to play. The player progresses in both dimensions simultaneously, which makes it feel as if they are playing in two different but parallel worlds.

Methodyca's level design is a mix between the linear and hub-and-spoke models. The game has 3 overarching main Acts: the introduction to the story; the main body of the game; and the conclusion. The player travels in linear fashion from one act to the next, meaning that once the player reaches the next act, they cannot return to the previous one. The second act (the largest and main part of the game) consists of 4 areas the player can travel between back and forth. Figure 2 below illustrates a layout of Act 2 of the game.

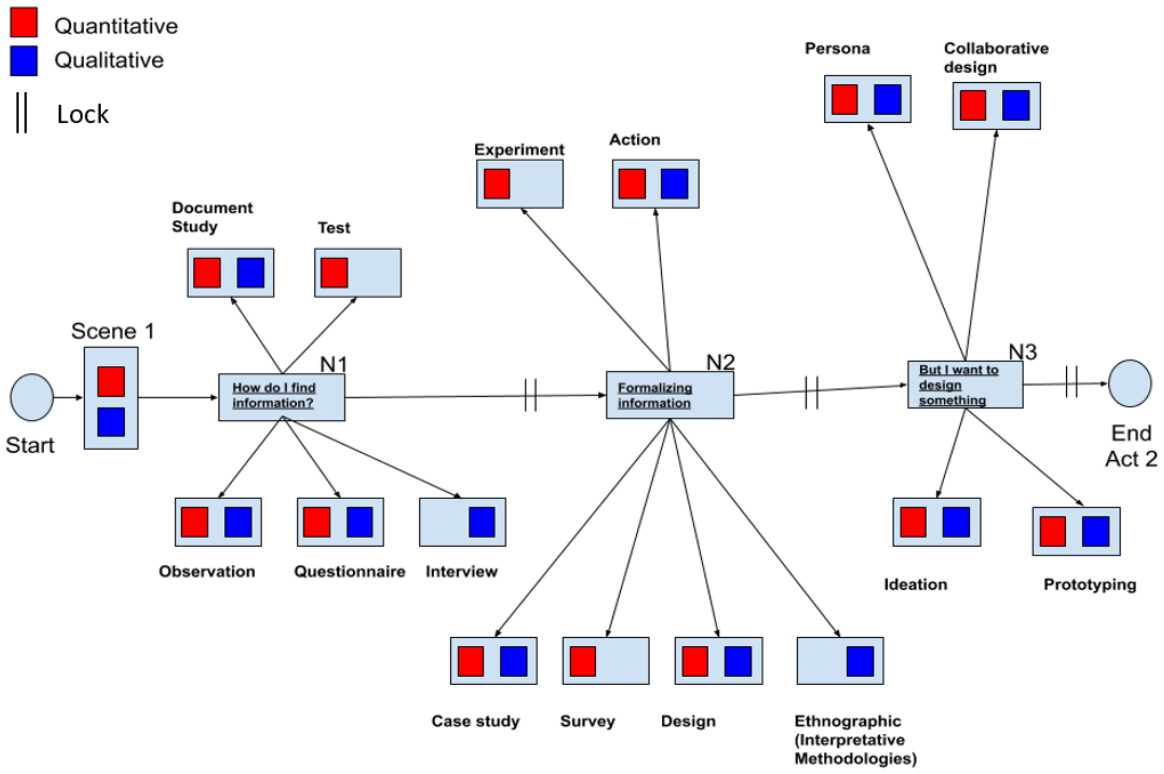


Figure 2: Methodyca Act 2 level layout (author: Peadar Callaghan)

As can be seen from the above figure, Act 2's four areas are named 'Scene 1', node 'N1', node 'N2', and node 'N3'. The nodes, and associated mini-games, are sequenced in a manner meant to take the player through the logical sequence of doing research, were node N1 represents the concept of "How do I find information?", N2 moves to the "Formalising Information" step, and N3 concludes with the showing the player the option of wanting to design a prototype or artefact for research. The '||' symbols along the arrows leading from one node to the next mean that access to the next node is blocked until the player has successfully completed any two of the mini-games connected to the node the player is currently at. The minigames that can be played in each of the nodes are linked to their respective nodes by arrows. Each minigame is defined by the type of research-related component it addresses, such as Document Study in node N1. The coloured squares at each mini-game indicate whether that research component is quantitative, qualitative, or both. These colours decide where a mini-game can be found, and accessed, by the player in either one of the two dimensions or both. Furthermore, each node

and its linked mini-games represent the hub-and-spoke level design model, where the player needs to be at a certain node in order to play a particular mini-game. From there, the player can only return to the same node before being able to move to the next or previous nodes, play another mini-game, or go to Act 3 after completing node N3.

At the moment of writing this document, Methodyca was not yet tested for game-play issues outside of the team making it. The public release available to play is the first version of the game (1.0.3) that a player should be able to complete from start to finish, as the designers and developers intended, without problem. A playable version of the game, and also its mini-games, is at the official Methodyca site (https://dlg.tlu.ee/methodyca/). Methodyca was designed and developed by a team of master students and instructors at Tallinn University. The team is comprised of:

- Martin Sillaots – Project Manager
- Mikhail Fiadotau – Narrative Designer
- Peadar Callaghan – Narrative Designer
- David Upshall – Game Designer
- Kaisa Norak – Game Designer
- Oluwafiyikewa Alawode – Sound Designer
- Ralph Söthe-Garnier – Web Designer
- Roman Gorislavski – Database Developer
- Anni Hõ – Game Artist
- Elena Gorshkova – Game Artist
- Iryna Selina – Game Artist
- Ahmed ElShenawy – Game Developer
- Sinan Emiroğlu – Game Developer

1.3. Development Methods and Tools

Nowadays, there are many options to choose from when wanting to build a digital game. Game engines, the set of tools used to create games, come in all shapes and sizes. Several programming languages can also be utilised to achieve the same results. For Methodyca, the

tool of choice was the Unity engine (https://unity.com) which has a free licence to create and publish game projects with. The choice of Unity over other free-to-use engines, like Unreal Engine 4, was because Unity is what the project developers are accustomed to. This way, the development team could start building the game right away, instead of having to understand how to use the engine first. Their familiarity with the engine also allowed them to streamline the process of preparing art, sound, and dialogue text files (assets) by the game artists, sound designer, and narrative designers respectively. The version of Unity used to make the current release of Methodyca is 2019.3. Creators of the Unity engine state that any version of the engine that is labelled as '**X.4**', where **X** is the year that the engine version came out (such as *2019*.4), means that this version has Long Term Support (LTS) (Unity Technologies, n.d.-b); which translates to 2 years of continuous fixes to the engine to remove any issues or bugs in it, and support for previous versions (X.1, X.2, and X.3) is dropped when an LTS version is released. As deduced from the previous statement, Methodyca was built using an earlier **X.3** version of the engine instead of an LTS version. This was due to the fact that development on Methodyca started around March 2020, while the first 2019 LTS version came out in June 2020 (Unity Technologies, n.d.-a). The team decided to keep working on the game using the 2019.3 version of Unity that development started with, until the first stable release of the game was made, before upgrading to the LTS version.

While Unity as a game engine is used to put together the game world and user interface using the art and sound assets provided by the artists, the programming of a game's functionality and mechanics is done in an Integrated Development Environment (IDE) software that is linked to the Unity game engine. Microsoft's Visual Studio Community 2019 was used as the IDE, because it's free, popular, and incorporates itself well with the Unity game engine. The programming language used to develop Methodyca is C#, also designed and developed by Microsoft, which may be another reason why Visual Studio is a good IDE to use with Unity. In fact, C# is the main and default scripting language that is used not only in Methodyca, but almost any game created using the Unity engine – 'almost', since older versions of Unity had UnityScript as an alternative to C# which was dropped in 2017 going forward (Fine, 2017). Test-Driven Development (TDD), and its guidelines created by Beck (2003), was a primary guideline

in building the main game's parts, where each new code functionality added was tested to see whether it worked as intended, and covered all use cases required by the game designer.

Source Control is an important part of any development project these days. It facilitates teamwork when making changes to the code, and it's great for making sure the code is backed up and any mistakes can be reversed. Furthermore, it helps the developers to go back and forth through the different development revisions of a product and see what changes were made and why. The development team made sure to use Source Control Management (SCM) from the beginning of the project. GitHub was the tool selected for the SCM. Similar to the reasons behind the other choices, GitHub offers a free source control service, in addition to providing GitHub desktop; a software that allows managing the source code using a visual and user-friendly interface, instead of command line tools. This allowed developers less experienced in source control and command line interfaces to still be able to manage their source code.

1.4. Process

Any digital game is composed of smaller systems that are combined together to form the final product. In a way, it is a manifestation of the phrase 'the whole is greater than the sum of its parts'. Methodyca is no different: it is composed of a collection of systems that connect and work together, where some are small and simple with one task to perform, and others are bigger, more complex, and sometimes cover the entire span of the game from the moment it opens till it is closed.

The systems in Methodyca which will be considered in this study are:

- World Navigation;
- Inventory System;
- Dialogue System;
- Badging System;
- State Saving and Loading System; and
- The Hub-and-Spoke Model: Integrating the mini-games.

Each of these listed items will be examined on its own in the following chapters. The analysis of each of those systems will be broken down into 3 main parts:

1. Theoretical overview and the design requirements that lead to the need for such a system and dictated the conditions forming it.
2. The development design process and outcome: How the system ended up being like, and how it integrates itself into the greater whole of the game.
3. Self-reflection and user feedback discussion: How effective is the system in delivering the design requirements and what is the player feedback concerning it.

Finally, a summary of the highlights of the study will be made, in addition to looking at some of the lessons learned and offering suggestions for improvement, which could benefit the overall experience.

1.5. Sample of the Study

The current COVID-19 restrictions made it challenging to have more in-person participants - and nowhere close to having the entire sample do the testing in person. This led to breaking the study sample into 2 groups: the first is formed by the participants who would test the game online from their own homes via website links, and answer an accompanying questionnaire during or after the play-test. The other group is smaller in number and comprises the participants who were able to perform the testing in person with myself. The live testing participants of the second group did not take part in the online questionnaire, since the questions in it could be answered directly by observing and talking to the participants themselves and while they played the game.

None of the participants of either group (online and live) played the game before the testing. Although the game was already released at the time of sending out the questionnaires and carrying out the live sessions, Methodyca was not yet openly disseminated.

1.5.1. Online Questionnaire Sample

Although the questionnaire was sent out and shared among Tallinn University students belonging in two class years of two different study programs, in order to reach as many

potential testers as possible, the response rate for the online questionnaires – which amounted to 3 results only – was unfortunately too little to warrant looking into.

1.5.2. Live Testing Sample

13 participants were asked to do live play-testing in my presence. This is how typical play-testing takes place in the game industry (*5 Things You Need to Know about Playtesting*, 2019), where the game testers play a version (or build) of the game, and the designers, developers, or both are also present. Play-testing is an important step in the development life-cycle of a game for designers and developers alike (Shin et al., 2020), to help them validate that players, essentially the end users, are playing the game how it's meant to be played and experienced (Choi et al., 2016), and to confirm any errors in a game's systems (or as developers call them: bugs). A post-play interview is also usually held between the development team and the game testers to record additional feedback regarding the test session which wasn't noticed through observations alone – this was the same with Methodyca's sample that did the live testing session.

Of the live testing participants, 54% of them were of the age range between 20 and 30. All 13 participants were university students or graduates. As for their experience with research methods; 4 stated that they know and actively apply research methods either in their work or studies, while 4 are only familiar with research methods but have not actively used them, and the remaining 5 did not know about research methods. The participants were also asked if they played video games before. To that, 5 said they play video games regularly, 2 play video games occasionally, while 6 either rarely play video games or not at all. About 30% of the participants were both at least familiar with research methods and also played video games either regularly or occasionally.

It is not critical, for the sake of this study, that the participants must be professional researchers, since what is examined is the game's functionality and playability itself. However, it does help if they know about research methods to be able to go through the entire experience with little problem, and that the game's design elements don't interfere with testing its functionality. Likewise, participants need not be avid gamers to be able to understand the

game and figure out whether they're stuck because of a designed challenge or a game error. After all, Methodyca is designed to be as simple and direct as possible to play with a minimal set of controls (a mouse is essentially all that's needed to play it).

1.6. Data Collection

To review Methodyca's systems, the game's source code and project files are required. The code and files are readily available and do not need to be collected by any special means for this study, since these discussed systems were developed by myself. As mentioned before, the source code is also available to the public on the online GitHub repository.

The questionnaires were created and accessed by participants using Google Forms service. This allowed quickly setting up a digital questionnaire and collecting data and statistics efficiently, as Google Forms already provides counts of numbers of answers, and creates response pie-charts for some questions. The questionnaire comprises a combination of closed-ended questions, for example 'Did you see a video cut-scene play after you used the activated portal?'; Multiple-choice questions, like 'Which mini-games did you play in the first area?'; and open-ended questions, such as 'Please mention if you had any problems or bugs with the inventory and using items in it'. The complete questionnaire, and its accompanying documents, are available in **Appendix A**.

In the case of the live testing sample, the questions in the online questionnaire sample were answered by simply observing how the participants were playing the game and interacting with it. Notes were taken for any issues/bugs occurring during game play, in addition to useful remarks about playing habits or actions performed by the participants. A short post-test discussion was held with the participants to gain additional feedback and comments on the overall experience. The observations and feedback were recorded in a spreadsheet, where a copy of it is available in **Appendix B**. Live sessions are considered more useful than the online questionnaires; where the questionnaires are done solely on the discretion of the participants, the live sessions on the other hand allow the developer to observe subtle player actions, whom may either forget about or dismiss as unimportant to report.

To focus the testing process and collected data on Methodyca's systems and functionality, and to avoid having the participants being stuck at points in the game due to encounters they might deem too challenging to progress through (which is design rather than development-related) and become unable to test the entirety of the game; special versions of the game were made specifically for testing purposes, where all the mini-games were allowed to be won without requiring the participants to complete them, by clicking an additional, test-version only, button in the mini-games' starting menus. This was optional, and participants were still free to play the mini-games until the end and win them by normal means if they wished to do so. The reason behind this is that the mini-games are separate from Methodyca's main systems that are being examined here, except the world navigation and state saving systems. Nonetheless, the design and functionality of the mini-games are extraneous and the only component that links them to the aforementioned two systems is either winning the mini-games or exiting them at any point, which is still triggered by the added buttons that simply skip the stretch of the mini-games and limit the testing to Methodyca's main systems.

1.7. Data Analysis

The retrospective study itself is a qualitative analysis of how the system is performing in regards to what is required of it. As mentioned above, this document serves as the reflect and disseminate step in the RtD research project, and self-reflection on the design choices and lessons learned from building the systems that make up Methodyca encapsulates the 'reflect' part of the 'reflect and disseminate' notion. The self-reflection analysis was carried out by looking back at the requirements set by the game designers, and then the design choices made to code the game's systems that leveraged these requirements and realized them in the game. An examination of how the earlier revisions and prototypes of the systems grew and changed: what parts remained and which were dropped or transformed. In addition to self-made notes, lists of issues to tackle, and bug reports from internal testing prior to game release; all studied and reasoned to form the knowledge that results from this research effort.

Responses to the questionnaire were planned to be analyzed by descriptive statistics, to for example understand the likelihood of certain situations occurring or not. This would have

indicated the appropriateness of Methodyca's systems from the end user's perspective as opposed to that of the developers. Open-ended questions were added to allow participants to provide any additional thoughts that the developers might have missed in the questionnaire. As for the live testing sessions, the data was recorded into a spreadsheet, and both the observations and feedback were grouped by participant and separated as points. A simple code was used to group the different observations and suggestions by the systems which they were associated with.

The live sessions were mainly qualitative, where observations and feedback of each tester were recorded and compared against the entire live testing group to check for dominant topics among the participants. Testers' answers during the post-test interviews were written down on the spot.

2. POINT-&-CLICK GAMES

"An adventure game is an interactive story about a protagonist character who is played by the player. Storytelling and exploration are essential elements of the game. Puzzle solving and conceptual challenges make up the majority of the gameplay. Combat, economic management, and action challenges are reduced or non-existent." (Adams, 2010).

Graphical adventure games are one of the oldest game genres to exist since the early 80's, and used to include hit game titles (Moss, 2011). In 1985, the game *Déjà Vu* was released on Apple's Macintosh computer, and was the first game to offer a 'fully point-and-click interface' (Moss, 2011). Déjà Vu introduced separate windows to model the game world, the navigation system, inventory, actions the player could take, and game narration.

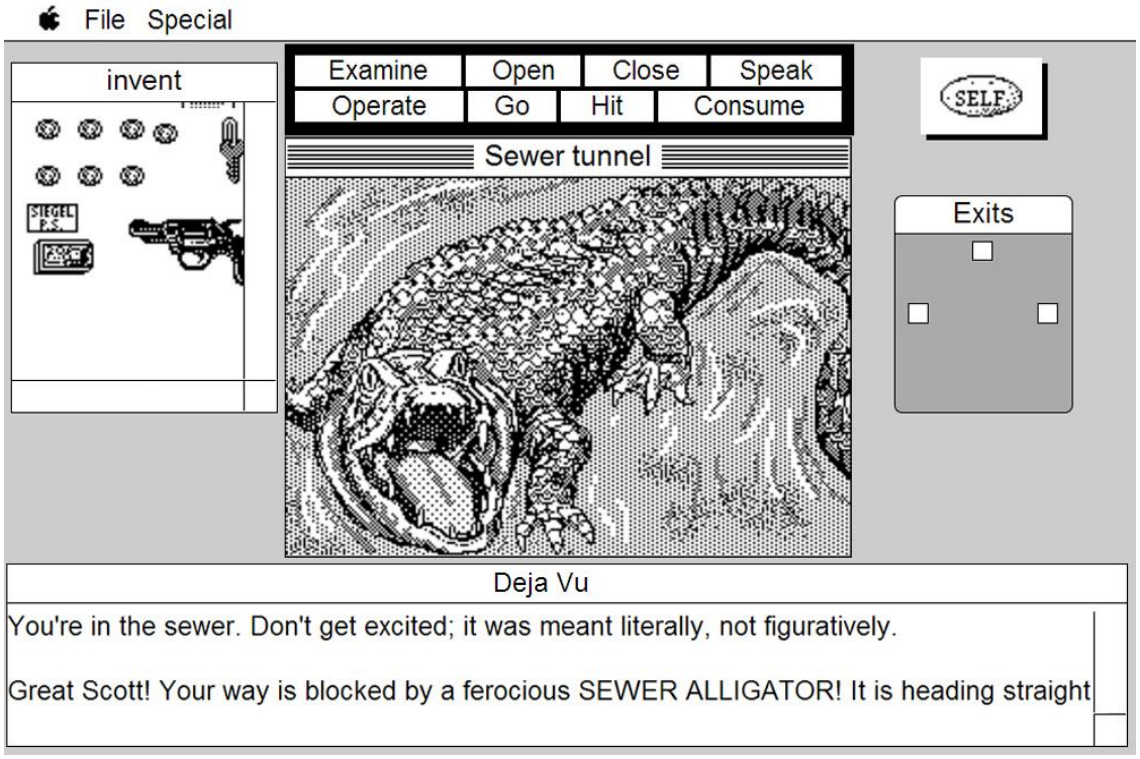


Figure 3: Screenshot from Déjà Vu (ICOM Simulations, 1985)

Through the years, the genre grew and more games were created like The Secret of Monkey Island (Lucasarts, 1990), Broken Sword (Revolution Software, 1996). Even nowadays, this style of games is still relevant in the games industry and modern point-and-click games are being

made. Some popular titles, according to G2A (2020), are The Journey Down (Skygoblin, 2010), Deponia (Daedalic Entertainment GmbH, 2012), 2017's Ken Follett's The Pillars of the Earth (Daedalic Entertainment GmbH, 2017), and one of the most recent examples is The Longing (STUDIO SEUFZ, 2020). Modern games in the genre, like these mentioned examples, tinkered with various art styles, narratives, and puzzles. Yet, all share in heart the roots of their predecessors.

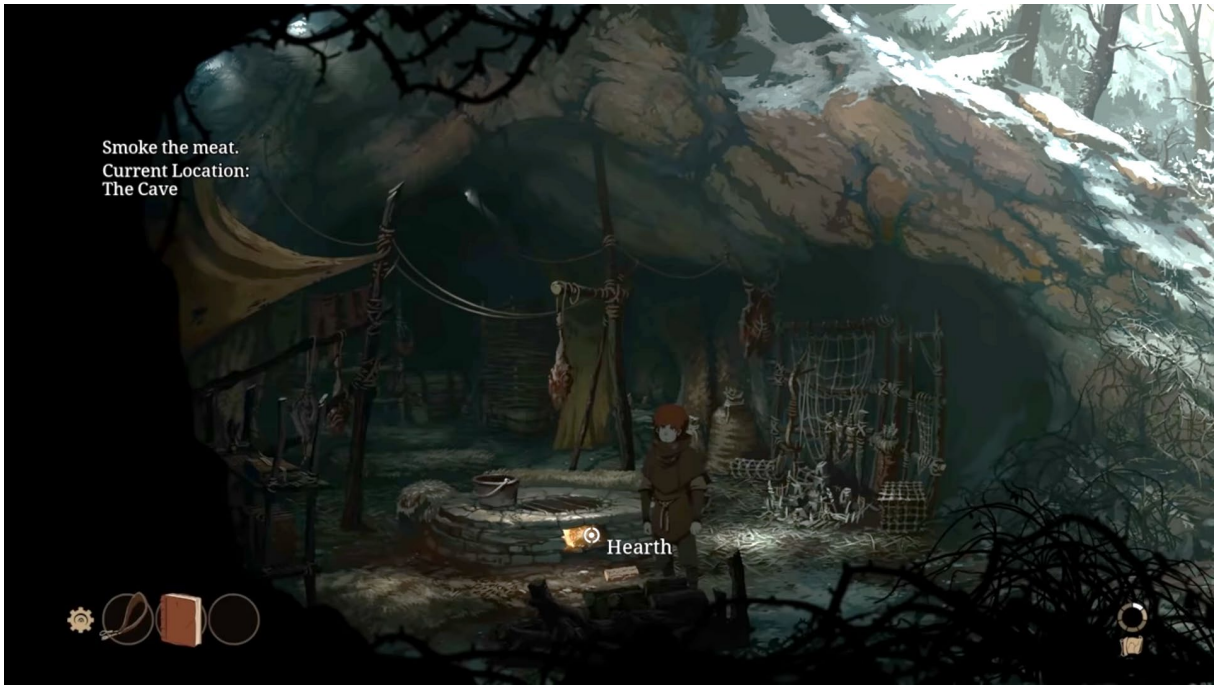


Figure 4: Screenshot from The Pillars of the Earth (Daedalic Entertainment GmbH, 2017). The mouse cursor (white circle) is placed on the hearth to use the wood log, from the inventory, on it – the log image is under the cursor indicating this action.

Although these types of game have different presentation and art styles and can be either 2D, 3D, or a mix of both as in the game Syberia (Microïds, 2002), they all have several common aspects and features that make them belong to the point-and-click genre: the player can navigate through the game world by clicking on objects or locations displayed on the screen; the player has an inventory to display the items they pick up from the world, and can access at a later time; the games are mainly based on puzzle solving to progress through them; there's either a narrative or a dialogue system to learn about the game's story and/or next objectives; and, most of all, using the mouse to 'point and click' is almost always the primary method of

interacting with and using these features. Of course, there can – and most likely will – be more elements or features in different point-and-click adventure games, but these listed ones are the main building blocks found in them all.

Granted, we have an idea what features and mechanics to consider if we wished to make a point-and-click style game. However, from a developer's point of view, the question persists: *How* to build such features? And *how* to generalize and modularize them, so that they can be used over and over to create an expansive game world?

3. WORLD NAVIGATION & INTERACTION

The first step in building any game is deciding whether the game world will be traversable or viewed entirely through a static viewpoint – like in card games. Since the focus here is on adventure games, there isn't much of a sense of adventure if the entire game world is seen and played in one single spot. Hence, travelling through the world is mandatory.

3.1. Theoretical Overview & Design Requirements

As already stated, any conventional point-and-click adventure game brands the ability to traverse the game world and interact with it, typically using the mouse to do so. Games like The Pillars of the Earth (Daedalic Entertainment GmbH, 2017), shown in Figure 4 above, had the player directly clicking on the spots where they wanted their in-game character to walk to, use, look at, pick up, etc. This kind of mechanic makes all relevant actions contained within the game world and evokes the feeling that the player is directly manipulating it. Other games like The Sea Will Claim Everything (Kyratzes, 2016), Figure 5 below, are more similar to the old-school style of the genre, where the main window is a viewport to the game world that puts the player in first-person view, and clicking on objects/characters in it is to inspect or interact with them. In contrast, travelling and pivoting is achieved by clicking on buttons in a separate window that are available where a movement action is. The main advantage of this style is that navigating the game world is always clear to the player, especially if the viewport is cluttered with many paths: if there's no button to a specific direction, it means we can't go or look there. Although visually and design-wise the two games' navigation and interaction mechanics look and act differently, as far as development is concerned the systems are very similar: when the player is allowed to go somewhere, they will have a button (either in game world or in a window) to click on for that.

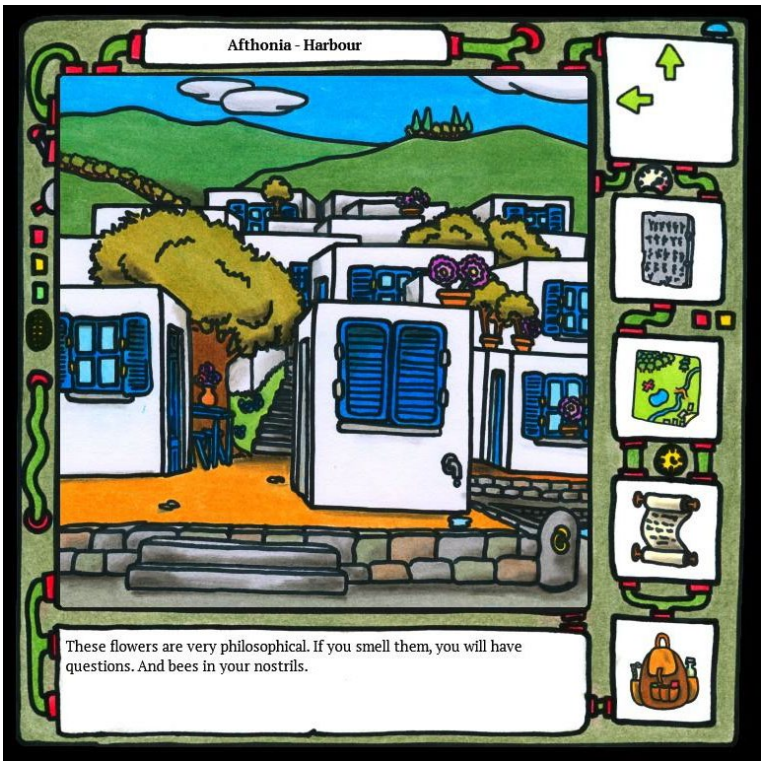


Figure 5: Screenshot from The Sea Will Claim Everything (Kyratzes, 2016). Green arrows in top right corner window display what movement actions are allowed; going forward or turning left at this point in the game.

The Methodyca designers opted to go with a mix of the old-school first-person view and the Pillars of Earth style of world navigation and interaction: The player views the world through their character's eyes and would click directly on specific spots in the game world to move from one location to the next, and click on the screen's right or left edges to rotate and look around at their surroundings. Some locations let the player turn around and even back-track their way to the beginning of the level (the second level), while others were restricted to only one direction to look at – as in the first level. The player would also directly click on objects and non-player characters (NPCs) to interact with them by looking at, using, picking up, or speaking to them. An additional twist in Methodyca is the ability to teleport between 2 dimensions at any location in the second level, using a device the player has in their inventory. This added another layer of complexity to how the world is traversed.

3.2. Development Process & Outcome

Although this study is not about working with a specific game engine, but generalizing the concepts to apply them anywhere, sometimes the unique aspects a certain game engine

provides must be considered as they can influence the design choices of the developer when building a system or a feature. In this particular case, the Unity engine is made to create both 3D and 2D games, and its User Interface (UI) functionality is powerful and flexible enough to even allow a developer to build an entire (but most likely small) game using only Unity's UI if they wished. Therefore, the first choice was in which perspective is the game world to be built in.

Methodyca was always planned to be viewed and played through a 2D perspective, so a developer might choose to build its world as a 2D game or use the UI interface if the engine allowed it. Actually, using Unity's UI to drive the world navigation and interaction was considered for Methodyca's world and navigation at first; as it makes developing the point and click mechanics fairly easy and straight-forward. However, this was quickly dismissed because Methodyca had UI elements that needed to be laid on top of the world view. Things like dialogue and inventory windows, and in-game menus were all going to be naturally made in the UI part for the game, and having to deal with building a game and its menus all bunched together on top of each other was going to be a nuisance in the long run – and fertile ground for program bugs. The next viable option was building it as a 2D game; since all the game art is 2D this made sense to use. On the other hand, this also raised the issue of laying out the different game screens. If all the locations in the game were single screens, it would've worked: simply move the player from one location/screen to the next, either by switching out the backgrounds and objects in front of the camera, or by relocating the camera itself to face the already placed-but-static locations. In Methodyca, however, some locations were like rooms with 4 walls and the player was supposed to rotate and see all 4 sides of the room, to simulate the feeling of standing there and having things on your sides and behind you – emphasized by the first-person perspective the player has of the game world. It would still be doable; all one-sided and 4-sided locations could have been placed and organized in a way to make it work as intended, and it wouldn't have been a wrong or mistaken choice to build the game scenes in this way. Nonetheless, this method felt like a burden as organizing the scenes in this way would've been time consuming. In addition to the fact that in a team of more than 1 developer, one's own view of an organized level structure is likely to differ from another's and thus

confuse the rest if they attempt to work with it. More importantly, this method didn't help in making the game world modular: the intention was to make each location a small piece that could be added to the next like building blocks, allowing the developer to make any given level as small as one room or as big as a hundred, in a streamlined fashion that followed the player's progression flow.

With those considerations in mind, it was decided to take advantage of the 3rd dimension to make the player 'walk' from one location, or room as they are called in Methodyca's development lingo, to the other. The player doesn't see themselves actually walking between rooms, but instead they are instantly transported inside them. Each room consists of 4 'walls' facing North, South, East, and West, and includes parameters to control how the player character moves in it; such as whether the player is forced to face a specific direction each time they entered the room, or whether they're allowed to rotate in it and by how many degrees. The turning angle degrees in Methodyca's case were either 90 degrees for a 4-sided room, or 180 degrees for a 2-sided one. However, this parameter gives the possibility to also make 6-sided or 8-sided rooms or more if the designer and developer wanted to. This room template acted as the building module for all the locations in Methodyca, and each level became a modular assembly of rooms placed in the same logical order as the player was expected to traverse the world in, hence simplifying the layout of each level and the structure of each room/location (see Figure 6). A room where the player won't be allowed to rotate simply had its 'player turning' parameter locked, and only the side the player will always face needed to be set up with graphics and interactions, while other sides were left in their default placeholder states.

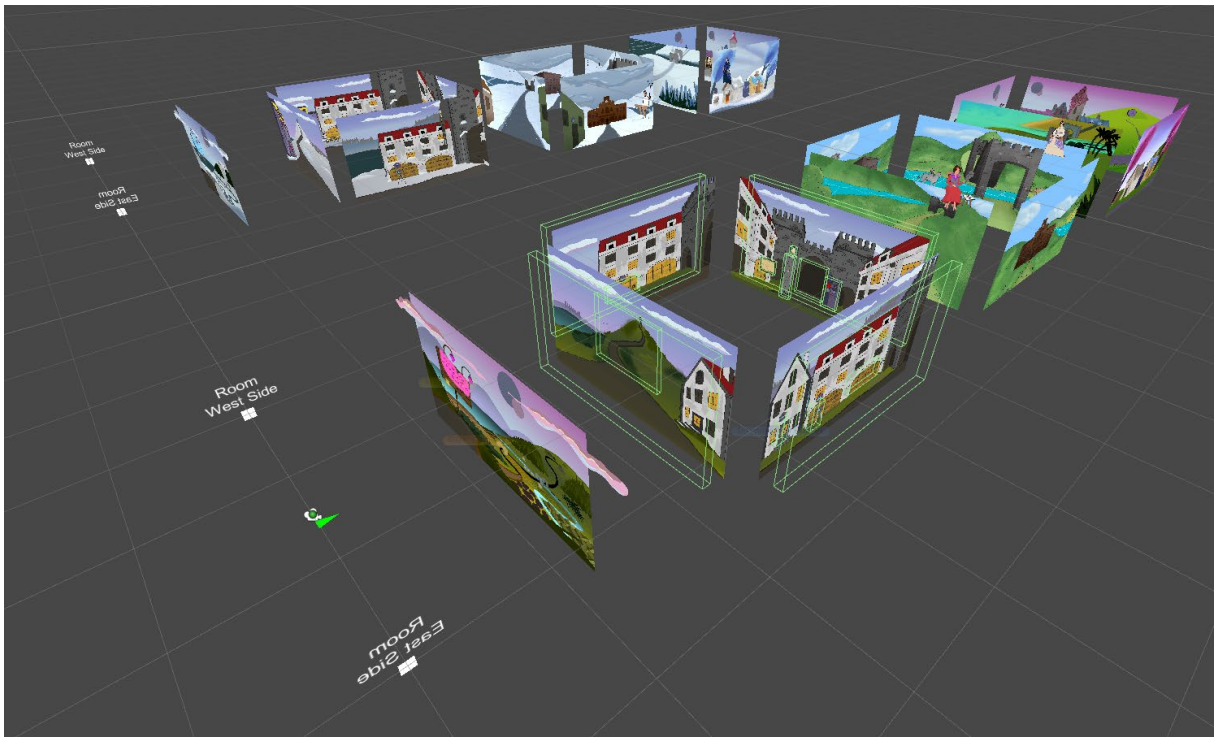


Figure 6: Rooms layout in the second level (Act 2) of Methodyca. There are 6 four-sided rooms and 2 one-sided ones with placeholders on either side. Green arrow represents the player's location.

The only layout organization done was that the spacing between each room was the same – arbitrary – value of 45 units. There was no requirement to do this and it was simply to tidy up the level, yet this little extra work was quite useful in creating the dimension-switching mechanic in the game.

Applying the same approach of making elements reusable, the objects and NPCs in the game which the player interacted with were all derived from one base class: Interactable Object. The interactable object class would include the main common parameters in anything the player can interact with, such as whether it can be interacted with at any given moment during game-play, the in-game description if the player chooses to inspect it, and whether it requires an item (and what is the item) to be used with it. From this interactable object parent class, child classes emerged that had these common parameters in addition to their own unique ones that distinguished them from each other. These child classes would become the templates that make all the interactable objects in the game, and define how an object reacts when the player interacts with it. At the start, these templates covered the basic interactions found in point-

and-click games like inspecting, using, picking up, going through (doors), or talking to an object or NPC. Some objects, though, required more complex interactions than these basic ones – such as the lock box in the office in the first level of the game. Such objects needed to have their own custom-made code scripts written for them to function as intended. Nevertheless, with these 5 simple classes acting as templates, a lot of the game's interactions were already set. Then, it was simply a matter of switching out one graphic with another to represent different objects and NPCs.

Figure 7: Different types of interactable objects represented by labels. Available interactions from left: inspect (map), speak (NPC), go through (door), and use (toll machine).

3.3. Reflection, User Feedback, & Discussion

From an introspective point of view, having a room template for each location in the game proved quite useful in quickly putting together the game levels. Creating the template itself needed some thought and general considerations, but after that it was a matter of making and placing duplicates of it, replacing graphics, and adding objects in it. With the hierarchical organization most (if not all) game engines provide, it was easy and quick to tell in which side of which room did an object or NPC exist. One of the very first issues that faced the method of

placing rooms next to each other in 3D space, though, was that the ray-casting from the player's mouse on the screen, to determine what object is interactable/clickable, would register objects in neighboring rooms when the current room the player's in contained nothing to intercept the ray-cast. There are several solutions to this, simplest of all is to limit the distance of the ray-cast; so as not to travel too far beyond the boundaries of the current room and cause this undesirable effect. The solution that was adopted in developing this system was to: a) Create an invisible collider object on each of the room's sides, that lies behind all the objects in the room, to intercept the ray-cast. This served the other purpose of controlling the cursor image among other things; and b) Disable all the rooms in a level but the one the player is standing inside. This not only ensured no objects in other rooms could intercept the ray-cast, but also meant to reduce the game's processing overhead by disabling the runtime code in all the objects in other rooms which the player cannot see and won't interact with yet. Although untested, this technique is expected to allow the developer to put many more rooms per level without it slowing down, than not having them be disabled.

As for the Interactable Object parent class system for interacting with game objects, this design started out well when considering the limited number of interactions first expected in the game. But as more unique objects and interactions were required in later areas (like portals, minigame links, etc.), it became apparent that this design wasn't the most flexible even though it got the job done eventually. A lot of more code needed to be added to child classes to serve these unique interactable objects, and very little of the parent Interactable Object class was used. One possible alternative to how this system is implemented, if the programming language allows it, is to use interfaces instead of inheritance to only use the necessary object interaction functionalities – luckily, Unity's C# scripting language does let the developer use interfaces.

Even though the world navigation worked very well during the play-testing sessions, the interaction with objects was, unsurprisingly, not without its bugs. Some usable objects displayed their negative feedback messages, which are supposed to show if they can't be used, after the fact of being operated by the player. Fortunately, this issue isn't a game-breaking one, but may confuse the player and should be fixed. The feedback received from participants of the

play-testing sessions included points relevant to both the design and the development of Methodyca. The observations and feedback points related to world navigation and interaction with objects were filtered from the live session data to be associated with the current topic.

The observations about world navigation and interaction were:

- Cabinet in supervisor's office displayed a feedback message saying ‘can't open cabinet’ comes up after normally clicking and opening it – 4 occurrences.
- Toll machine displayed negative feedback on correct use – 10 occurrences.
- Player couldn’t tell where to click to start a minigame. Example: having hard time finding where to click to start observation minigame.
- Player clicked the minigame icons, thinking it will start minigames.
- Player clicked on the empty table in front of Monster NPC (instead of clicking the NPC) to give them the game board.
- Player took some time to notice the 'Return' button on the lock box interface. Instead, clicks outside the graphic to exit or clicks the lock box handle after unlocking it – 2 occurrences.
- Player attempted to click on the gate after using the toll machine to open it, instead of clicking the toll machine itself.

The participants’ feedback on interaction elements were:

- Enjoyed the puzzle-solving part in the supervisor's office to find the crystals.
- Would like to be able to switch the left mouse button and right mouse button controls.
- Cursor images pointing right and left on screen edges for turning could be bigger.
- Suggested highlighting interactable objects.
- Suggested adding sound feedback along with the visual cue when hovering over interactable objects to help near-sighted players.
- Reported a bug: The map next to Map Maker NPC shows an interaction, but nothing happens when clicking it.

- Would like some feedback on what is clickable/interactable in the main game. When asked about circle on cursor, replied that they didn't notice it.
- Liked the circle on the cursor to indicate interactable objects.
- Had no clue that they can turn around in 2nd level (Act 2), and would like a hint about it.

As can be seen from the results, most of the feedback is more about the design rather than development bugs and problems. This is expected, as the participants were simply asked to play the game and report their findings, and development issues aren't as easy to notice as design ones. Nevertheless, implementing any design changes based on participants' feedback automatically translates to development work, which should be tested after completion for any possible new bugs coming about as a result. The feedback concerning the possibility to switch mouse controls is considered development-related, since it does not really affect the design of the game and is a matter of player preference. For this specific matter, it is recommended that if this suggestion is to be implemented, then it can be an option in the game's settings that is left up to each player to decide what is more comfortable to them.

4. INVENTORY SYSTEM

4.1. Theoretical Overview & Design Requirements

Adventure games often rely on puzzle solving for the player to progress through the game. Many of these puzzles involve the player using or combining items together for something to happen. Point-and-click, as a type of adventure games, also have these mechanics in them. An inventory system is the feature which supports these kinds of item-related puzzles, as it allows the player to pick up and hold onto an item from one location, and then have access to it and use it at another location. With this, a puzzle is no longer forced to have both itself and its solution together in the same screen, and allows a game designer to add more complexity to solving it.

Inventories in point-and-click games come in many shapes and sizes, but they generally fall into two categories of usability: either always visible and accessible along with the game world; or available in a separate window that the player has to open, select an item from, and then close the window to return to the game world (and use the item in it). The first category can be seen in games like The Pillars of the Earth (Daedalic Entertainment GmbH, 2017) in Figure 4 above and Broken Sword (Revolution Software, 1996) in Figure 8 below. The inventory is usually situated in one edge of the screen, typically the top or bottom as they tend to be longer than the right/left edges and thus can fit more items on screen at any given time. The advantage of this style is that both inventory and game world are visible at all times and directly accessible while the player can see events taking place in front of them instead of being cut off from them to use the inventory. However, this method also limits the number of items the player can carry in their inventory as it needs to fit the screen width.

Figure 8: Screenshot from Broken Sword (Revolution Software, 1996). Inventory items are displayed in the grey bar at the top of the screen.

The second category is illustrated in The Sea Will Claim Everything (Kyratzes, 2016) in Figure 9 below, where the player's inventory is accessed by clicking the backpack icon in the bottom right corner of the screen. Once there, the game world is replaced with a list of the items the player is holding and can be inspected and interacted with in multiple ways – in this particular example, the player can touch, smell, eat, or ask a wise little mouse about the item. This approach to the inventory system is like a mirror to the first; it solves the disadvantage of limited space by offering a list that can hold any number of items, but suffers from hiding the game world from the player whenever they want to check the inventory. The choice of which system style to use in a game will depend on the designer's needs and priorities.

In Methodyca, the first style of inventory system was chosen – essentially looking the same as the Broken Sword one. The player wasn't expected to carry a lot of items and the levels weren't large. Hence, having items displayed in a bar at the top of the screen was enough for the game's requirements. Design team discussions took place about how the player should also use the items in their inventory. For example, should the same items (coins for example) stack on

each other, or placed next to each other in the inventory bar. Another point of discussion was whether the item needed to be picked and added to the inventory in the first place, if the object that required it was also in same view of the player – making it so that the player only needs to click-and-drag the item directly towards the object and skipping the inventory. However, it was eventually decided that this doesn't lie in the spirit of point-and-click games. Additionally, mixing between draggable items and inventory items would only create confusion to the player about how they should expect to deal with them.

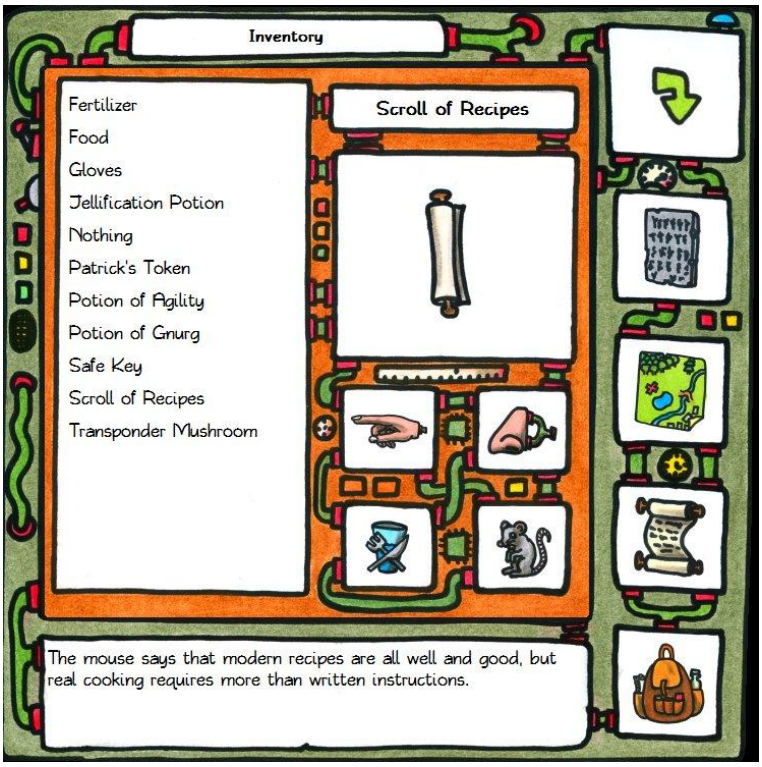


Figure 9: Screenshot from The Sea Will Claim Everything (Jonas Kyratzes, 2016). Inventory window shows a list of items the player is holding, with one item 'Scroll of Recipes' being inspected.

4.2. Development Process & Outcome

The inventory system was built using an online video tutorial about creating inventory UI for role-playing games by Brackeys (2017) and was then expanded upon to fit Methodyca's needs. Conceptually, the inventory system is divided into 3 parts: Inventory Manager; Inventory UI, and Inventory Slots. Figure 10 illustrates the setup of the system.

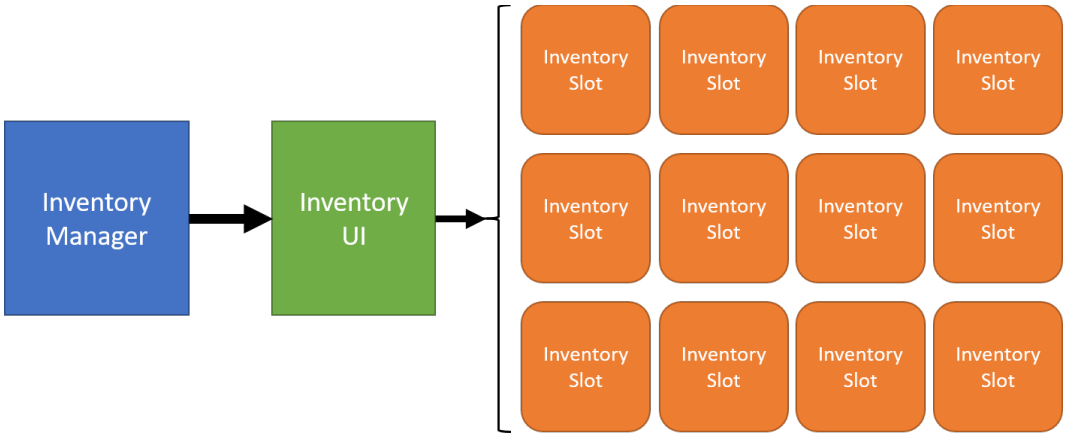


Figure 10: Inventory system scheme

The inventory manager comprises the main system code which contains a list of all the items the player is carrying, sets the limits of how many items can be carried, and handles adding and removing items from the list of items the player holds. The inventory UI is notified each time the inventory manager adds or removes an item from its list, and handles displaying and hiding inventory slots based on what items are in the manager's list (hence carried by the player), in addition to setting the graphic of each carried item. The inventory slot is a template object that is duplicated as many times as the carry limit of the inventory manager, and is responsible for two things: displaying the items the player currently is holding in their list of inventory items, being a visual representation for the player; and acting as a button, which when clicked on lets the player carry the selected item to use it in the game world. In Methodyca, the maximum number of items which can be carried, and can fit on the screen, is 12 (plus 2 more slots reserved for the game menu and dimension-switching device). This number can be increased, however, if the size of each item slot is decreased to fit more slots. This inventory system is highly flexible due to its simple design, and depending on a game's needs it can fit many more items by simply adding more inventory slots; either at one edge of the screen like in Methodyca, or in a separate window if the developer and designer wish to adopt a style similar to Jonas Kyratzes' (2016) The Sea Will Claim Everything.

4.3. Reflection, User Feedback, & Discussion

As mentioned above, the beauty of this system is its simple yet expandable design which can fit many situations. In fact, the tutorial it was based on didn't even make the inventory system for

a point-and-click game but a role-playing one instead, which are usually much larger in size and scope than the point-and-click genre games. Although 12 items are more than enough for Methodyca's needs, this limit might be too little to other games that plan for the player to carry more items most of the time. One solution was already mentioned in that the slot sizes can be reduced to fit more, but this can only take the developer so far. If a significant number of item slots is required, then perhaps other means of displaying the player's inventory should be considered. Luckily, this means only the last part of the system will need to be adjusted, while the inventory manager and inventory UI parts will more or less remain the same – this flexibility is the main selling point of this system.

The way items are taken from the inventory was programmed in such a way that the player should hover to the top of the screen to display the (normally) hidden inventory bar. Then, by clicking on an item, it would be moved to the player's 'hand', which is represented by transforming the cursor graphic to be that of the item, so that they can use the item. During the play-testing sessions, it was observed that some testers would attempt to click and drag the item from the inventory bar to where they wanted to use it, only to feel frustrated by the failed attempt and having to go back and only click on the item. This is perhaps more of a design-related matter; however, it is a valid control mechanic which warrants looking into to streamline player interactions.

5. DIALOGUE SYSTEM

5.1. Theoretical Overview & Design Requirements

Story and narrative make a large part of adventure point-and-click games, and motivates players to play the game and figure out its story (Marchiori et al., 2012). In adventure games, story is typically provided through dialogue with characters, and the player is given options to choose from when it's their turn to talk. All the games mentioned so far in this chapter, even the 1985 Déjà Vu (ICOM Simulations, 1985), contain NPCs that the player can speak to. Some of these examples which worked by clicking on command buttons, like Déjà Vu and The Secret of Monkey Island (Lucasarts, 1990), had a button just for speaking to characters (and objects). This demonstrates how much narrative and dialogue is important in games of this genre. Since Methodyca was planned to be a point-and-click adventure, having NPCs and a dialogue was a natural decision. The work on the character dialogues was planned to be split between the development team, building a dialogue system with choices for the player to pick from, and the narrative design team, working on designing the different characters and writing the actual dialogue pieces for each NPC the player encounters.

5.2. Development Process & Outcome

At first, the dialogues and all in-game messages were going to be displayed to the player using a 'dialogue manager', which was to be programmed by the development team. The dialogue manager did well for displaying small single and multiple messages, but that was only the case for simple pieces of text like those appearing when inspecting an object or using an item. However, when considering the more complex dialogue trees with interweaving sections and multiple choices for the player to choose what to ask or discuss with a character, the burden quickly multiplied. Around half-way through the project, it became obvious that building a system which would support character dialogues in a proper fashion will require a large amount of development hours, not to mention working out a workflow between how the narrative designers should create and export their written pieces to the developers to include in the game.

With the need to finish the rest of the game, it was not a good idea to put in so much time to develop such a system. The narrative and development team started looking for an alternative solution, which was found in Inkle's freely available 'ink' scripting language and Inky: a tool to create and edit textual story and dialogues using the ink language. This tool was already proven to work by its maker Inkle, as they are game developers themselves who used this technology to create their own narrative-based adventure games such as Heaven's Vault (Inkle, 2019), 80 Days (Inkle, 2014), and Sorcery! (Inkle, 2013). Narrative designers would learn to use Inkle and ink scripting language, write up the dialogues and test them within Inkle, then pass over the files to the development team, which used a plugin created by Inkle for the Unity game engine to read these ink files and embed them in the game files. The plugin also contained a code library to tap into for coding the dialogue and dialogue choices. With that, the dialogue obstacle was solved for the development team, and both narrative designers and developers had one common and proven tool to work together with.

As mentioned in the previous section, all interactable objects including NPCs were based on the parent Interactable Object class. Modifying the NPC child class to fit in all the new changes that came with the Inkle plugin seemed like it would tear away the NPC class from the family of interactable objects and make it its own thing. Furthermore, not all NPCs in the game functioned exactly the same by just displaying dialogues; each NPC needed to check the player's progress in the area they were standing at, and some NPCs would even give the player and take from them items that progress the game. Coding all these variations in one single NPC class was completely unintuitive and against object-oriented programming (OOP) principles. In that regard, additional scripts were created that were unique to every NPC and fit with their requirements, while all of them belonged to a parent class that shared the main common elements involved in every dialogue created with Ink, regardless of the NPCs' distinctive needs. Interacting with an NPC interactable object would then trigger the accompanying new script created to employ the Inkle plugin and dialogue functionalities.

In the end, both the simple dialogue system developed in-house and the Inkle plugin remained in the final release of the game. The simple system was kept to display texts for generic

gameplay messages such as picking up or using items, and the Inkle plugin was used solely for the dialogues with the NPCs. Figure 11 below shows an example of either system in action.

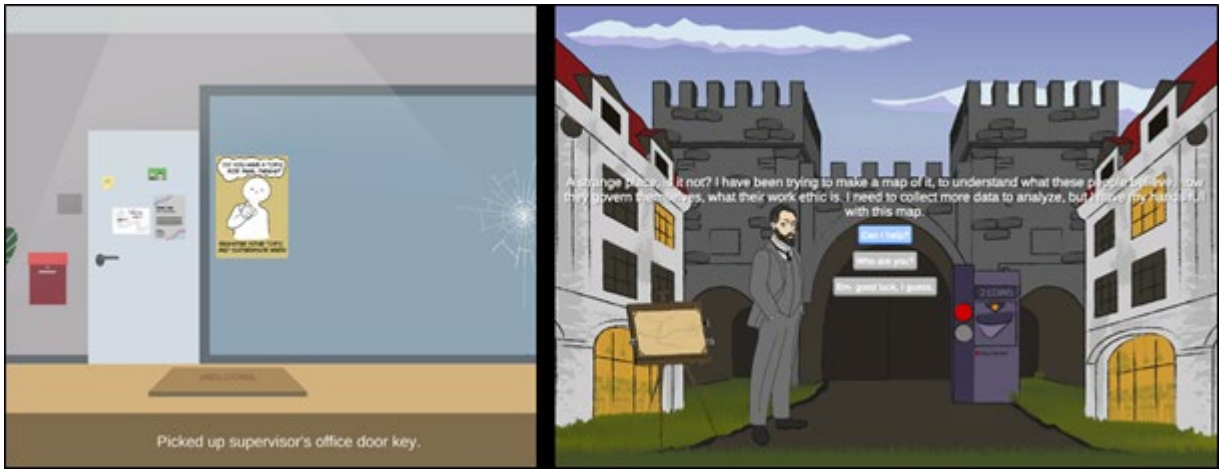


Figure 11: Left screenshot showing the simple dialogue system, and right shows the Inkle dialogue

5.3. Reflection, User Feedback, & Discussion

In hindsight, it was an error on behalf of the development team to assume a dialogue system is a simple or easy task, without carefully recognizing all of its aspects – including the writer-developer work and data flow, which is an important bit to figure out. Making a simple system that displays some messages is straightforward, but when having to account for dialogue trees, linking them to in-game variables, and writers with different styles and word editing software preferences: things can become chaotic quite fast. It is imperative for the developers and writers to sit down early on in the project lifetime, and understand what is expected in the final product, which determines whether it's feasible to develop a custom-tailored system. The (simple) dialogue system that the development team started on may have worked, if it were further developed to meet the requirements of dealing with branching dialogues. However, the development team consisted of only 2 members, and building something with such complexity is a small project in itself and is not a prudent choice for a short-term project or a small development team. Besides, even if the system was completed, there was the other matter of planning and standardizing how narrative designers should create and share their write ups, in a manner that can be understood and used properly by developers.

Using third-party tools and plugins like ink/Inkle greatly helped in cutting down the development time and worry. It also standardized the writing and dialogue importing process for all narrative designers; as they all had to abide by the rules of the ink scripting language, thus helping the developers in quickly understanding the flow of the text regardless of the designers' different writing styles. On the other hand, a plugin is not a magic bullet that automatically solves all problems. The time to understand how to use and work with a tool should be considered, specially that plugins are usually generic and need to be customized according to a project's needs. As the ink-supported dialogue system was introduced fairly late in the project life and almost right before the release date, there were several features that had to be dropped (such as having close-up portraits of NPCs showing different facial expressions depending on the dialogue) due to not being familiar enough with the plugin and running out of time. Furthermore, the narrative designers were not coders, and for some the ink scripting language was their first time with something of the sort. This led to a few bugs turning up when implementing their dialogue files into the game code. One example was that the Inkle editor would display the last line of text an NPC says before ending the conversation, but in the game the dialogue window disappears at that last line and the player cannot see it and thus become confused as to the sudden break in the communication. Another example was that some narrative designers added dialogue choices in the Inkle editor at the first step of their designed dialogue tree to help them go check the different conversation paths they set up. When the files were placed into the game, these options, which were meant to be automatically picked by game code, were the first thing a player sees – this was considered a dialogue bug. Although fixing those 2 issues was fairly simple, these examples showed that files submitted by the narrative designers still needed to be reviewed by the developers and not taken for granted.

With the first released version of the game, there now exists two dialogue systems: the version created by the development team and the Inkle plugin. It would be wiser to transition completely to a single dialogue system rather than keeping both; as this creates needless work overhead of having to debug and improve two systems that aim for the same goal.

During the play-testing sessions, both dialogue systems worked fine in their own terms. Regardless, one of the NPCs, Map Maker, had a recurring bug where on the player meeting him for the first time and choosing the first dialogue option, the conversation would abruptly break and players were no longer able to interact with or click anything in the game or the UI. After investigating the bug, it was found that the dialogue file written by the narrative designer was missing a code command that broke the dialogue from continuing and led to the bug. Once fixed, the conversation functioned normally. This points to the aforementioned remark of having the developers review the narrative designers' submissions and not taking the possibility of errors for granted. Additionally, some interactable objects like the toll machine displayed the negative message meant to show up when the player couldn't use the object, but instead it came up after testers used the toll machines the right way – the message shouldn't have been displayed then. A suggestion from one of the participants was that the current way of how the feedback messages is displayed after an item is used (to confirm to the player that it's been used) should not block the player's interaction by forcing them to click a second time on the screen just to dismiss the confirmation dialogue. Instead, it would allow the player to continue interacting with the world, and the dialogue message can disappear automatically.

6. BADGING SYSTEM

Methodyca's design team decided to include badges in it, mainly as a way for the player to keep track of their progression within the game, as Zichermann & Cunningham (2011) put it, and to know which minigames they completed and which they might like to try out. Naturally, if Methodyca's players decided to use its badges to compare their progression and achievements with one another, it would be a useful motivator to play more research-related minigames and maximize the use of Methodyca as a learning game.

6.1. Theoretical Overview & Design Requirements

A badge is "a validated indicator of accomplishment, skill, quality or interest" (MacArthur Foundation, 2013). Game badges, or achievements, are rewards the player wins in a game when they perform certain feats in it. TrueAchievements is one website that lists and tracks the achievements for the Xbox game console games, with additional guides on how to obtain them. Badges are a tool which game makers use to get players to engage with their games and products outside the games themselves, and are also one method of measuring players' progression within a game (Zichermann & Cunningham, 2011).

Badges have become widely used in games in recent years; specially after being implemented by gaming platforms such as Microsoft's Xbox Live service and the Steam online game store (Hamari & Eranti, 2011). Early games like The Secret of Monkey Island (Lucasarts, 1990) and Broken Sword (Revolution Software, 1996) didn't have any of those, and the entire experience was self-contained within the game, without interest in rewards outside the game experience itself (Jakobsson, 2009). Conversely, when Lucasarts published a re-mastered version (updated graphics and sound) of their original game 20 years later under the title The Secret of Monkey Island: Special Edition (Lucasarts, 2009), they did introduce achievements to keep up with the recent trends. While badges and achievements are typically not part of the core game experience and have no impact on it (Jakobsson, 2009), Hamari and Eranti (2011) argue that players' efforts to collect them makes them far from being a secondary activity. Some games even leverage those player efforts by hiding some achievements without any description of

how the player would go on collecting them; leaving it completely to the player's guess, dedication, and luck to find out about them – these are often called on the internet as secret achievements.

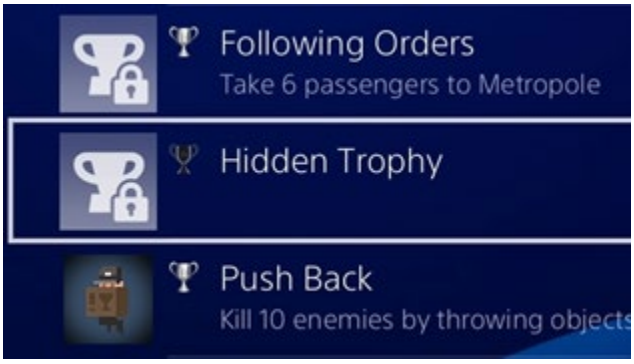


Figure 12: A secret achievement on the PlayStation 4 games console is named 'Hidden Trophy'

In Methodyca, the badges for the minigames are accessible from the player's (in-game) mobile device in the inventory bar, which is essentially the in-game menu. The player can see all the badges available, but they're all locked at the start of the game. The player then needs to finish the minigames to unlock their corresponding badges. There are no secret or hidden badges, and the description for each badge explains directly which minigame is required to win it. The design intention behind this was to keep the player aware of their progress through the game, and if they left the game and returned to it at a later time, they could remember what minigames they played.

6.2. Development Process & Outcome

As the badges are tied to winning the minigames, and since minigames are distributed not only throughout the different locations in a level but also across levels, a badge system was created to handle all these specifics. The badge system was made based on the inventory system mentioned above, to take advantage of the same flexibility and modularity the inventory system has. In other words, the badge system is also made up of 3 main parts: Badge Manager, Badge UI controller, and Badge Slots. The badge manager has a list which keeps track of all the badges won, and the badge UI script is notified every time the badge manager updates its list. The process goes in the following sequence: the player completes a minigame; the badge manager checks and confirms that a new minigame is completed, thereby adding that

minigame's ID code to its list of completed minigames; the badge UI is notified that a new badge is earned, finds the associated badge slot and unlocks it for the player to see in the Badges menu. How the badge manager checks completed minigames will be addressed in the next chapter.

The first difference between the badge and the inventory system is that the player starts from zero/no badges and keeps collecting them, but never loses any badges that were already collected. This means that the list of badges earned only increases as the game progresses, while the list of items in the inventory manager can increase and decrease in size as the player goes through the game. The second difference plays on the fact that because the earned badges list only grows, and each minigame is unique, then we can assign the minigames ID numbers – which makes it easier to keep track of which minigame is which across the program code in a clear and concise manner, that is less prone to typing mistakes, as opposed to having to write the minigames' names every time they needed to be referenced. This case is not possible, nor desirable, with the inventory system/items because: a) Some items are duplicates, like the coins and punch cards; and b) There is no real need or value to giving items ID codes. In contrast, the same minigame is referenced in code several times; from badges, to giving item rewards, to saving and loading, and more. Of course, using numbers as IDs may sound a confusing option as well because the developer will need to remember all the IDs. This is not the case however, as the IDs are created using an enumeration, which offers the double benefit in a game engine like Unity in that it translates the enumerations into a drop-down list to select from - Figure 13 shows this in action.

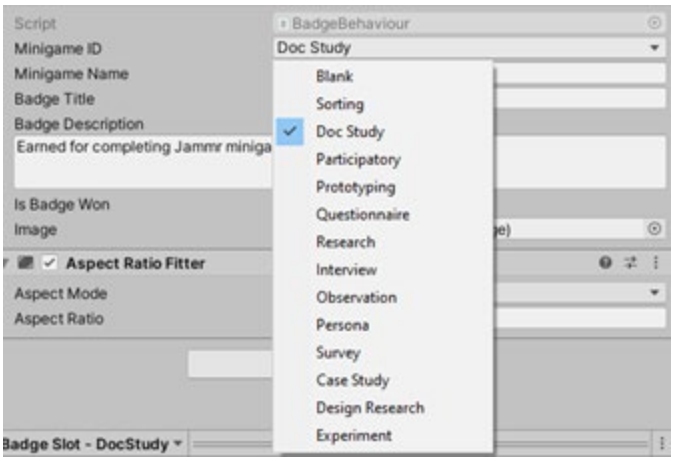


Figure 13: The 'Minigame ID' parameter inside the Unity Editor, with 'Doc Study' selected. Since the parameter is from an enumerator, Unity translates it into a drop-down list menu instead of manually typing in the value.

Enumerations automatically translate a name value to an integer value, which becomes that name's reference in the list, in other words its ID. Meaning the developer doesn't need to bother with coming up with ID numbers and ensuring they're not already being used. For example, 'Sorting' in the list in Figure 13 above is the name given to the Data Charger minigame in the IDs list. Since it's the second item in the enumeration list after 'Blank', it is automatically assigned the numerical value 1. The next item 'Doc Study' is assigned 2, and so on. This method makes dealing with multiple elements (13 in this case) easy and efficient. At any point, should the developer need to add another minigame to the list, they simply add it to the end of the collection, and it shows up in the Unity editor, and can also be referenced in code via its number value or by the IDE's autocomplete functionality. The only complication that a developer needs to be aware of is to avoid adding a new item somewhere other than the end of the enumeration, which pushes the remaining items (and their references/IDs) by an increment of 1. This can be very problematic if those earlier elements were already referenced either in code or in the game engine, as this basically means all those references will all need to be updated, otherwise forgetting only one reference can either create a bug or crash the game if used in something sensitive to its base functionality.

6.3. Reflection, User Feedback, & Discussion

The way the badge system is built proved to be quite flexible, modular, and adapting to expansion. Much like the inventory system. The choice to use an enumeration to deal with the

minigames in terms of numbered IDs was also fruitful in integrating this system with the state saving system, which is discussed in the next section, as having to save only numbered values on file. The only drawback is that once the enumeration collection is tied to other parts in the game code, rearranging the list becomes troublesome for the same reasons mentioned above. If the elements of an enumeration are not too many and not used extensively, this may not pose a problem. However, when a system grows large, then updating references may cause issues if a couple of them were forgotten. This issue can be avoided early on by careful planning and identification of the elements which populate an enumeration; thus, reducing the need to fix or change things later in the development process.

In terms of play-testing feedback, there was none. One hypothesis is that the entire game was quickly completed in a single session, and the participants had no opportunity to look at the badges or discuss them over with others. It is expected that players outside a testing environment, who will take their time playing the game and discussing it with others, would induce more feedback and discussions about the badge system and badges. Regardless, when checked at the end of each play session, all participants have earned the correct badges and no bugs were found with the system.

7. STATE SAVING & LOADING SYSTEM

State saving, or game saving, is when a game saves the player's progress up to a certain point, either automatically or manually, so that the player can stop the game and upon returning to it, they can pick up where they left off. Game saving is meant to allow players to leave the game without losing their progress or having to play the game from the very beginning every time (Sirlin, 2008).

7.1. Theoretical Overview & Design Requirements

Early game consoles like the Nintendo Entertainment System, or NES for short, didn't support game saving, as the console didn't even contain hardware capable of doing so (Robbs, n.d.). Their games used password systems, which when entered by the player the game would reconstruct the save state based on the password's characters (Imagine Media, 1996). Newer games, like God of War II (Santa Monica Studio, 2007), let the player save their game at specific save points while also including hidden checkpoints for the player to continue from while the game is running (Sirlin, 2008). With even newer, more capable hardware, games became able to combine both the manual save points and hidden checkpoints together, so that the player could go through the game without worrying about losing their progress, or at least losing as little as possible if they chose not to or forgot to save their game before stopping it – Gears of War 2 (Epic Games, 2008). Nowadays, almost any game made by a professional development studio has a save system that records player's progress continually; unless it was intentional by game design, such as permanent death (Bartle, 2004).

With Methodyca consisting of 3 levels and 9 different minigames, it was obvious that players may not be able to complete the entire game in one sitting, nor were they expected to do so. Having a way to save progress and return to it later ensured that playing the game can be more comfortable and convenient to the player to play in their own time and on their own terms – something that Sirlin (2008) attempted to prove as a positive thing for the game. In this regard, a system which allowed the game to be conveniently saved and loaded, in addition to auto-saving checkpoints, was necessary. Furthermore, having the player go to separate minigame

levels, and returning to the main game world to carry on where they left off already required something not too far from a save system; to restore the game world, object interactions, and inventory items to their previous states. Methodyca needed a system similar to Epic Games' (2008) Gears of War 2, in that it combined both saving and checkpoints together.

7.2. Development Process & Outcome

Unlike saving a game while playing it from a player's perspective, saving game information on the development end isn't something that can be done by simply selecting what we want to save, press a button, and magically have all the information stored in a save file. Saving and loading game states and information is its own system like any other; one that requires planning, selecting the right pieces of information, and choosing the right method to store this information (Uccello, 2017).

Taking the planning step as the foundation for the game saving and loading system, it was necessary to understand what elements, or variables, in the game were static and what were dynamic. Static and dynamic does not mean whether objects will be physically moving in front of the player's eyes, but instead whether the information that define these objects are expected to change as the game is played; for example, the office key hidden under the doormat at the start of the game, which the player has to pick up in order to unlock the office door with. The office key is a dynamic object, in that at one point during gameplay it is there and at another it's gone (when it's picked up). The office door is another 'state-dynamic' object; although it doesn't change visually, its state changes during gameplay from locked to unlocked, when the player uses the key with it. On the other hand, an object like the sticky note inside the office (first level) that tips the players about the lock box's code, would be considered a static object from the state point of view; because despite the sticky note being an object the player interacts with just like the key and the door, its state never changes during gameplay – the note will always trigger the same feedback text when clicked will never move nor disappear no matter what. With this understanding in mind, a simple list consisting of what *types* of objects were expected to change their states during the game was made. The list mainly covered interactable objects such as those that are moved, picked up, used, locked and unlocked, and

NPC interactions. Additionally, the save system also needed to know where in a level was the player standing, to return them to the same spot upon loading. In the same manner, both the inventory and badge managers needed to save what items the player is holding and badges they earned respectively. Also, when loading a game state, it is necessary to know what minigames were already completed, so that the player isn't forced to replay them every time they load their game (or abuse the game by replaying the same minigame to collect the items needed to progress).

After evaluating which objects and systems needed to save their states, the next step was to analyze what pieces of information in each of those elements needed to be stored and recalled, so that when a player loads their game, the entire scene is reconstructed to how it was like when they left it. It doesn't make sense to save every program variable in every object, nor is it even necessary to recreate the scene – a few smart and informed choices are enough to get the job done. Looking back at the examples of the office key and door objects: the key is a pick-up type of object and the door is, unremarkably, a door type. As the key object is only meant to be picked up from the game world and stored in the player's inventory, a single variable is the only information that needs to be stored; is the key picked up or not? If the key is picked, then the next time the level is loaded that key will be removed from it. In the same manner, the door is only meant to be unlocked (if it was originally locked), and thus again only one variable needed to be stored; is the door locked? If the stored information upon a level load says no, then the door will be automatically unlocked. From the player's perspective, everything will just look and act the way they left it the last time they were playing – and this is exactly what the developer needs from a save system. Of course, doors and pickable objects are two simple examples of what interaction information requires saving. The portal object at the end of the first level, which is the gateway to level 2, is a more complex example. The player needs to place 2 crystals on it, which means the portal can have a state of either no crystal placed, only the purple crystal placed, only the blue crystal placed, or both crystals placed on it – 4 different states compared to 2 in the case of the key or the door. To translate these states into easy-to-store data, each state was given a number: 0 for no crystals, 1 for purple crystal, 2 for blue crystal, and 3 for both. When the level loads, the portal object looks up the saved number value and

from it decides which state it will be in. In hindsight, this seems like a fairly straight-forward matter, but reaching this solution is anything but a straight path. The developer needs to analyze every unique object in the game, and anticipate each possible state that object can be in when the player might decide to save and leave the game. Only then can they devise an idea of how to compare these different states in a meaningful way. For the inventory manager, the name of each item the player carried in it was saved as a string value to be re-added to the inventory when the game was loaded. This also meant that each item needed to have a unique name, so that the inventory manager can tell them apart and recall the correct ones. To illustrate this, in the second level where the player earns 2 coins from playing minigames, although both coins look and behave in the same way, they actually have unique names in the game code: 'N1 Coin 1' and 'N1 Coin 2'. The way the inventory bar and manager worked in Methodyca was that each inventory item was considered a unique one, and occupied its own inventory slot. Were both coins just been named 'Coin' and the two were still in the player's inventory before saving and exiting the game, the inventory manager would have been confused when the game is loaded; it needs to re-add an object named 'Coin', but is it one object or two? For a program, it will just add one coin and stop, never asking this question or wondering if there ever was a second object of the same name. In the same manner, the badge manager used unique information to save which badges were earned. Since the minigames already had individual number IDs linked to them, as explained in the above badges section, those IDs had the added benefit of acting as that unique bit of information which needed to be stored and recalled upon game load. The last bit of information is related to placing the player in the same location they were in when a level is loaded. Normally, when a scene/level loads, the player always finds themselves at the spot that the developer set as a starting point – usually the beginning of the level. But when loading a saved game, the player expects to find themselves where they were the last time. And if the game uses specific save points in the levels, like in God of War II (Santa Monica Studio, 2007), then they naturally want their character to be at the same save point they saved at. In the case of known save points, the game only needs to know which was the save point and spawn the player character at its location. However, if the player has the ability to save their progress anywhere in a level, then it

is most likely needed to store the player's position in 2D or 3D space depending on the game. Since in Methodyca players could save their game anywhere, storing information about their 3D location was the way to go. The only challenge though, was that the game world was built as a set of rooms, and the Game Manager responsible for handling the world navigation disabled (effectively hiding) all the rooms except the first room the player starts at by default. This meant that spawning the player character at their last saved location, without also re-enabling the room they were in, made it look like the player found themselves lost in a stark black world of nothingness. Thus, instead of using the player's location information, the name of the room they were standing in was stored – so the game would enable the room and then spawn the player inside it. Consequently, this meant that every room in the game also needed to have its unique name.

The final step in building the state saving system is to establish how all the aforementioned information would be saved in a streamlined manner. To this end, a Save Manager code script was written to act as: a) A data storage container for the combined information which was stored by all the in-game objects as described above; b) A handler for serializing, storing, and retrieving the saved data files on disk; and c) A data organizer and provider, for the game objects to fetch information about their states. The save manager needed to be available at all times during the game's runtime, for any object to record its state right away. Thus, the save manager script was made to exist as a code class which lived outside the game scenes and thus was independent of where the player was in the game or which level was loaded. Thus, any object in the game could essentially reference the save manager and request to store or retrieve information about its state. When the player saved their progress through the game menu, the save manager would copy all the aggregated information to a file and save that file in the web browser's cache on the hard disk drive. The saved file consisted of: a state number, which corresponded to either the auto-save or one of the other 3 manual save slots; name of the level the player was in; name of the room the player was in; a list of the names of all the items the player had in their inventory; a list of all the names of the objects the player interacted with and their state of interaction, defined as true/false values; and finally, a list of all the IDs of the minigames the player completed. Since Methodyca was developed to be

played online using web browsers, reducing the size of the save file would ensure that downloading and saving it on the player's machine would be as efficient and unobtrusive to the gameplay experience as possible. To achieve a small file size, all the objects' states were adapted into either true/false (or 0/1) splits, or numbered values – such as the portal and its 4 crystal placement states example. Likewise, where names needed to be stored, as in the case with the inventory items, the names were all kept as short as possible. When writing a file to disk, every number and character in that file adds to its total size. By keeping the names short and describing states in single digit numbers, the save files were no larger than a couple of kilobytes. As a reference, a save file from The Secret of Monkey Island: Special Edition (Lucasarts, 2009) is between 5 and 13 kilobytes in size (PC Savegames, 2018).

To create the auto-saving functionality, which entails that the game automatically saves the player's progress rather than relying solely on them manually saving it, each room in a level had a parameter to determine whether the room was a checkpoint or not. If the room was set as a checkpoint, then every time the player enters that room, the game would automatically save their progress up to that point. The auto-save slot in the game menu differs from the manual slots, in that the player cannot control or manually save on it like they do with the other slots – as it is controlled by the save manager and the way the developer setup each room in the game. Since Methodyca has a lot of minigames to play, and the developers not wanting the player to risk losing their progress and having to replay them, almost all the rooms and locations in the game act as auto-saving checkpoints. On the other hand, the design and development teams decided not to allow players to save their progress inside the minigames themselves. This is because of how complicated game saving is from the development perspective, and the diversity of the minigames meant each of them needed a custom solution. Additionally, each of the minigames was short and required not more than 15-20 minutes to complete, which made the efforts to add a save functionality unfeasible. Instead, it was deemed satisfactory to automatically save their progress once players returned to the main game.

It should be noted that these design choices were also constrained by the Unity game engine itself, and made based on the developer's experience in it. In other words, the developer should be considerate of the tool they're using and examine if it allows other options and possibilities to handle the situation.

7.3. Reflection, User Feedback, & Discussion

The game saving system was arguably the biggest and most complex system in the game. It took a long time between designing, building, and testing it to make sure everything works as expected between multiple game saves and loads. As this was the first time for the development team to ever create a state saving system, almost all the work done on it was based on their and the designers' personal expectations and experience from playing other games. Subsequently, it became obvious very quickly into the system's development that if a game is to have a save system, then it is something that needs to be planned from the early days of the project, as integration with other game systems is key to its success. The save manager itself was quite clear in its adaptation; keeping lists and saving them to file. The real frustrations arose when several objects and systems all attempted to load their information at the same time, while also being dependent on each other. If object A depended on object B, and object B loaded its data *after* object A, this meant that object A basically remained in its default state and did not update itself based on how object B actually is. As a game engine, Unity runs the code for all scene objects in a random order every time (Unity Forum, 2009), thus making it impossible to discern which object loads its data before the other. The solution was to modify the code in the game objects to ensure there's a proper sequence in loading their data and respecting the dependencies at the same time. Systems needed to load their state first while also prioritizing them by dependencies on one another, then interactable objects that may rely on these systems. Additional issues cropped up when saving was integrated with the minigames' levels and returning from them, which will be studied in the next chapter. With the save system being implemented after other systems were near completed (inventory, interactions, badges, and minigame links), integrating it into those systems was several times more challenging than if it were to be done in conjunction with them. Many bugs and issues kept coming up, and fixing them all became very time-consuming.

Even now, with Methodyca released, it would be good to revisit the code and make sure that all components which are dependent on each other are working in a more harmonious manner, as there is definitely room for further optimizations and improvements. In short, building an effective game save system is not something that should be taken lightly or for granted, and is something that should be considered carefully early on in a project's development lifetime.

One benefit that was recognized from this philosophy was that with state saving being its own system, changes and bug fixes in other parts of the game did not break the save files. As long as the data being saved for an object remained the same, the objects could still retrieve the saved data and function normally. This is unfortunately not always the case in video games and could be counted as an advantage towards the developed save system. In comparison, the recently released Cyberpunk 2077 (CD Projekt, 2020) had bugs which were fixed in a later patch, however, the patch also meant losing the previous saved games players had and they needed to restart the game entirely (Blake & Gerblick, 2021).

As was the case with the badge system, the save system hadn't received any feedback from the participants of the play-testing sessions; mainly because none of them tried, or needed, to save the game. Since the participants were asked to test the game from start to finish, and were also able to skip the minigames, the testing sessions weren't long enough to warrant the need to save progress and return to it. Therefore, despite the save system being thoroughly tested internally by the development and design teams of the game, it would still be beneficial to receive feedback in the future about its effectiveness from players which will be making use of it.

8. HUB-&-SPOKE MODEL INTEGRATION

Methodyca combines two level design layouts in its world design: linear with hub-and-spoke models. Figure 2 shows how the second level of Methodyca, Act 2, takes the player in linear fashion from start to the end of the level between the nodes marked N1, N2, and N3, while each of the nodes acts as a hub for the minigames linked to it. Even though the minigames are reached without the player needing to traverse more into the world, they still satisfy the fundamental roles of being the spokes of their respective nodes; since the player can only reach any minigame from its linked hub, and also must return to the hub to either play another minigame or move on to the next node. The previous chapter discussed how to build modular world areas without delving much into the minigames part. This chapter looks at how some of the game's systems, particularly world navigation and interaction, badging, and state saving and loading, have been designed to accommodate this style of level and game design.

8.1. Theoretical Overview & Design Requirements

Game levels can have many different layouts and combinations of layouts. In his book, Adams (2010) lists several fundamental level layouts such as linear, parallel, hub-and-spoke, and networks. Linear levels are just like the name says; one area or level comes after the next in a sequence which the player has no control over. This style of level layouts lends itself well to linear storytelling (Adams, 2010), where the designer wants to make sure that the player engages with every part of it in a specific manner. The downside to linear level layouts is the lack of replay value and surprises, since the player will always face the same events every time they play the game – unless they missed something before. The hub-and-spoke level layout entails a central area, the hub, which branches out into multiple other areas all connected to it; the spokes. Figure 14 below illustrates this form of level. The design of hub-and-spoke levels tends to have the player starting at the hub area, and then make their way to and back from each of the spokes. It is also possible to not make all the spokes available to reach from the start, by locking access to some of them so the designer ensures the player has to visit some particular ones first. However, the designer should note that locking all spokes but one, and

allowing the player access to the remaining spokes one at a time alters this into a linear level layout (Adams, 2010).

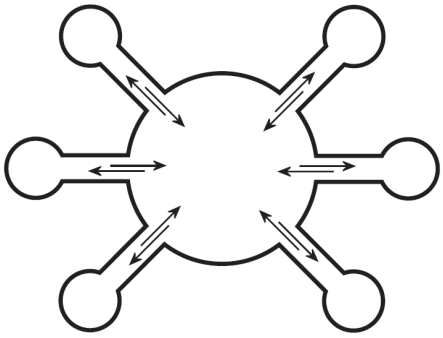


Figure 14: A simplified hub-and-spoke level layout (Adams, 2010)

Dark Souls (FromSoftware, Inc., 2011) is one game highly celebrated for many reasons, one of them being its level design (Ribbing & Melander, 2016). Its level layout has a combination of hub-and-spoke with linear tie-ins between the spokes that allow the player, when nearing the end of the game, to effectively travel between its spokes almost without ever needing to pass through its main hub; thus, transforming the level layout from a hub-and-spoke to a network-style one, where the different locations are accessible in multiple ways (Adams, 2010).

The hub-and-spoke level layout is not exclusive to point-and-click games in particular nor adventure games in general. Dark Souls (FromSoftware, Inc., 2011) is a mix of action, role-playing, and adventure genres, and it adopted a combination of hub-and-spoke with a network layout, in addition to a few linear parts (*Maps - Dark Souls*, 2019). Spyro the Dragon games (Insomniac Games, 1998) – a platformer genre – also exhibit hub-and-spoke level design, by placing the player in a 'Homeworld' that is the center world which links to other worlds in a game's universe (*Homeworlds*, n.d.). Not all hub-and-spoke levels need to look like the example in Figure 14 above. The map of Hollow Knight (Team Cherry, 2017) game follows a hub-and-spoke design, except that the hub is at the top of the world map instead of being the center of it, and the rest of the world expands and branches downwards into the depths – this can be seen in the interactive map of the game created by RainingChain et al. (n.d.). The top still acts as the starting position (hub) for the player and the place they most return to, specially early on in the game before unlocking shortcuts between some areas. And the areas under it usually

reach dead ends where the player either fights a boss, acquires an ability to access other new areas, or both; that makes them act as the spokes just like how Adams (2010) explained the principle in his book.

Methodyca didn't apply this model to explore an expansive world or universe, but the hub-and-spoke layout lent itself well to facilitate the designers' intentions with the game: The player needed to play some minigames and obtain some skills in research methods before being able to move on to the next section. Furthermore, minigames that shared common aspects in relation to research methods could be grouped around the same hub. The minigames accessible from node N1 in Act 2, for example, were all aimed to help the player understand and answer the question of 'How to find/collect information?'. The player isn't required to play all the minigames at a node, but needs only two in each node/hub to continue in the main game's world and beat the game. Nevertheless, players always have the freedom to play any number of minigames beyond the minimum requirement.

8.2. Development Process & Outcome

8.2.1. World Navigation & Interaction

The previous chapter examined how the main game's levels are built as rooms placed together in the order the player is expected to progress through the game. This covered the linear level layouts of the first and second levels - the third level doesn't count as it's made up of a single room. The minigames in Methodyca were developed separately from the main game, with each having their own mechanics and systems with little influence or connection to the main game. In fact, barely any of the systems made in the main game carried over to the minigames, and vice versa. Each minigame was designed and developed separately from the main game, and each was a unique product that looked and played entirely unlike the other. The disconnection the minigames had had freed them from any genre limitations as well; they were in no way required to play like adventure or point-and-click games (they only incidentally happened to all rely on the mouse as the main game controller). In other words, unique game designs commanded unique program code and scripts, ones that were made specifically for their

relative games and could fit none other. Therefore, the development team found it best to have those minigames as separate levels – or scenes as they are called in the Unity engine.

The parent interactable objects class acted as a base for another type of interaction, specially made for starting the minigames and aptly named ‘minigame interaction’. Just like usable and pickable objects or NPCs, the minigames were simply another type of object set in the game world which the player could interact with by clicking on it. The difference was that these objects were made to reference their respective minigame controllers and request to load that minigame when clicked on/ interacted with. Hence, the additional parameters that this ‘minigame interaction’ code had over the base interactable object were a reference to the required minigame controller (called ‘game hub’), a choice to what the interactable object should do when the minigame is completed and the player attempts to interact with it again, a Boolean to check whether the player can start the minigame or not, if the developer wished to lock access until a specific time, and a text response to display to the player when the minigame isn’t accessible.

Figure 15: 3 interactable objects leading to minigames with icons hinting at what the minigames are. From the left: Jammr (library window), Binoculars (binoculars tripod), and Interview Simulator (grey door).

The reason the minigame interaction was kept simple, and another controller for the minigames was made, was to stay true to the concept of modularity in game building and to decouple the logic of a main game object interaction and that of running minigames. All the minigames' connections needed to be through interactable objects, which is a notion that remained true regardless of those games' contents. Thus, this part was kept within the minigame interaction component. On the other hand, certain minigames required some preparation by the player before being able to play them; namely the 'Data Charger' and 'Research Papers, Please' minigames. Both minigames had some sort of interaction with the crystal items: The player cannot start Data Charger before placing 2 crystals on the desk where it is played; and both games needed to have charged up crystals available for the player to pick up once they were won. This level of complexity was beyond a simple object interaction to be covered in a single class of code. And so, a second, more specialized, script was created for each of these minigames to handle their specifics with the crystal items.

Even though they had a couple exclusive parameters, the 'game hub' minigame controller scripts did have common properties between them as well; just like the interactable objects. These common properties are applicable to all the minigames as well. These properties are: the name of the level containing the minigame, the scene loading type, and whether the minigame should be preloaded. The Unity game engine allows scenes to be either loaded singularly or additively, by either closing the current level and opening the new one, or loading the new one on top of the current one, respectively. Unity also allows scenes/levels to be preloaded, which implies that a preloaded level will be loaded in the background and not opened until the developer allows it either via code or gameplay trigger. Depending on the game and the developer's needs, one option or the other may come in handy. In fact, 'Data Charger' was one minigame that needed to make use of both functionalities. While the other minigames all played in their own independent levels and had loading screens – meaning preloading was not necessary, 'Data Charger' was meant to be a small minigame that is played within the context of the main game. Thus, it took advantage of those features by preloading the minigame as soon as the player entered the office, in order to keep loading time to a minimum when the player starts the minigame, and was also loaded on top of the main game – as it made no sense

to show a loading screen every time the player wanted to look at or away from a desk lying in the room they're standing in. Since none of the minigames besides 'Data Charger' and 'Research Papers, Please' had any unique requirements, a single 'game hub' code script was enough to be used with them all, which included the aforementioned common parameters, in addition to other parameters that applied to all minigames in Act 2 of the game regarding the simultaneous dimension gameplay – this is addressed in more detail in the next chapter.

8.2.2. Badging System

The main purpose of badges in Methodyca is to help players track their progress as to which minigames they completed and which are yet to be played and/or won. Therefore, it was crucial for the badge system and manager to be connected somehow to the minigames, and be able to discern whether a minigame was won or if the player just left the minigame without finishing it – to decide if they qualified for the game's badge. The challenge lies in that the main game's systems living in its scenes do not carry over to those of the minigames', as they were developed independently, which means that when the player starts a minigame, the entire badge system is left behind in the main game. Furthermore, when the player leaves a minigame, whether or not they completed it, and returns to the main game, the badge manager is reset to its default values and loses all information that it had, including information pertaining to previously-earned badges; This is due to the fact that the number and kinds of earned badges is dynamic information that disappears as soon as the badge system is removed from the active scene (minigames).

Fortunately, the Unity engine's code library includes a special feature for such cases – aptly named 'Don't Destroy On Load'. When utilized in a code script, the developer can force any object living in one game scene to never be removed from the entire game, even if the scene where the object originally existed is removed and another scene, or several, took its place after. This is a very powerful tool, but it also requires planning as it is inefficient to simply make everything in one scene move to other scenes, knowing that there's no real use for them elsewhere (such as level rooms in Methodyca's case). One part of the main game's code, the 'Scene Manager', had that option in it. The scene manager, as its name implies, is responsible

for handling the flow of the game's scenes/levels from the start menu, to the main game and minigames' levels, up until the end screen. Hence, it was one part of code that had to be omnipresent. This made it, along with the Sound Manager, two objects common between both main game and minigames – this fact was to be taken advantage of. A simple list was created in the scene manager object, similar to the one the badge manager uses to keep track of won minigames / earned badges, and was named 'minigames won'. Then, two separate script files were made: 'return to main game' script requests the scene manager to quit the minigame and return to the main game; while the 'win minigame' script adds the ID code of the current minigame to the 'minigames won' list appended to the scene manager. The implementation from this point was simple: Every minigame had a start game screen and a win screen for successfully completing the game, and both screens had a 'quit' button. Simply put, the start screen's quit button used the first script to only return to the main game – empty-handed – while the quit button at the win screen used both the first script to leave the minigame and the second script to save that minigame's ID code in the scene manager's 'minigames won' list. As soon as the player returned to the main game and the scene is loaded along with the badge manager, the badge manager would perform 2 actions: The first action involves retrieving stored information from the Save Manager regarding all previously won minigames (if any), to ensure it is up-to-date with the player's progress; and the second action involves checking the 'minigames won' list that the scene manager has, for any new minigames. This two-step process ensures that every time the player leaves any minigame, regardless of winning it or not, the badge manager will always keep up. The 'Data Charger' minigame was the only exception to the rule; since it's the only minigame in Methodyca to run on top of the main game as explained above – thus the badge manager was still available. To this end, the aforementioned 'win minigame' script was expanded to directly communicate with the badge manager, instead of the scene manager and its list.

In the same manner, the 'game hub' controllers on the interactable objects leading to the minigames benefited from the same approach. Upon returning to the main game, a game hub would also check the scene manager's list of minigames, and if it found a matching ID with its own minigame then it would either block the player from re-entering that minigame and

display a message confirming this, or completely remove the interaction-ability from the object – which is up to the designer/developer.

8.2.3. State Saving & Loading System

The saving system was briefly touched on in the badging system above, where the badge manager fetches information about completed minigames from the Save Manager. Loading minigames' levels not only disables and removes all the main games' running systems except scene and sound management, but it also resets all those removed systems and all object interactions when the player returns to the main game after exiting a minigame. As a result, the save manager plays a great and crucial role in maintaining player progression; not only to pick up from where they last left Methodyca off, but also to have a consistent overall gameplay experience between the game's worlds.

The previous chapter explained how the save manager code was developed to be independent of any level and be always accessible at any time. It is also unlike the scene and sound managers: while these two are still objects that reside within the game's levels, the save manager was created in the game's files and does not exist as an in-game object. This design choice was not made haphazardly, however. Simply put, the scene and sound managers are required to be objects existing within levels because they employ game-related code and deal with other game objects in the levels, which is much more convenient to do when the code lies within the same environment – just like the inventory and badge managers. On the other hand, the save manager didn't actually interact or deal with any game-related functionality per se. Its fundamental role was storing data during runtime, saving that data to disk, and retrieving it when needed; thus, there was no value or reason in operating it like other in-game systems.

In that regard, the save manager's persistence and auto-saving features were exploited for the smooth transition from main game to minigames and back. When the player is in the main game and interacts with an object that takes them to a minigame, before that minigame is loaded up, the progress until this point is saved in Methodyca's auto-save slot, which ensures that all object interactions, inventory items, and badges are saved on disk. Afterwards, upon

the player quitting a minigame to return to the main game, the save manager first recalls this latest auto-save file, then all objects and systems merely request their relevant data to restore their state to how it was before the player moved to a minigame and go on from there.

8.3. Reflection, User Feedback, & Discussion

8.3.1. Minigames as Spokes

Having the minigames be made separately from the main game allowed the designers and developers to work with complete freedom in realizing the vision of these minigames. Methodyca currently has 9 minigames (and 3 more in dialogue form), and not one of them looks or plays like the other. The hub-and-spoke model also lent itself well to making the distinctions between the art and play styles of the games justifiable without feeling jarring. In contrast, this complete freedom came at the cost of complicating how the minigames were joined into the main game, due to the absence of any common or standard pipeline to achieve this. Some effort was needed to understand and locate the correct places in each separate minigame for the links to the main game to be inserted – considering that each developer has their own distinct coding style as well. Fortunately, since the main game's systems were largely disconnected from the minigames, not a large amount of work was required to overcome this obstacle. Likewise, another issue that cropped up was that the developers of the minigames weren't aware of how their games were supposed to join into the main game, and so they didn't know what to look or care for during the development process of the minigames, which led to a few problems as well. One such example is game objects from one of the minigames carrying over to the main game on quitting, as its developers were also making use of the 'Don't Destroy On Load' feature. To counter this, an additional code script had to be explicitly written to filter out and remove those objects when leaving said minigame.

With regard to managing player access from the main game to the minigames, one could argue that splitting the code into a part for only interacting with the objects leading to minigames and another only for controlling their behavior – in the form of 'game hub' controller – is perhaps breaking it down into overly small fragments. However if, for example, the crystal handling logic

of the ‘Data Charger’ minigame is to be all placed in the same code script, then this ‘minigame interaction’ code could no longer be acknowledged by the generic term ‘minigame’, but instead something like ‘minigame with crystals interaction’ – in other terms, it loses its reusability (and modularity) advantage; as other minigames that don’t require crystals or any other prerequisite items prior to playing them should not be forced to deal with those parameters. Granted, this basically meant that a unique controller had to be created anyway to accommodate for the ‘Data Charger’ and ‘Research Papers, Please’ minigames. Conversely, this is arguably a smaller price to pay by creating a script that deals with a handful of unique elements, rather than duplicating the same entire base code a number of times for the sake of one or two additional parameters, which is ultimately counterproductive. Similarly, Robert C. Martin asserts in his book how smaller pieces of reusable code are more efficient, readable, and easier to maintain (R. C. Martin, 2008). Furthermore, what the game engine empowers a developer to do should be kept in mind when writing code scripts. Unity, Unreal Engine 4, and Godot are all game engine examples that let the developer attach multiple code scripts, as components, to game objects in order to modify and expand their behavior. Therefore, there is no reason as to why the developer shouldn’t take advantage of this to make smaller, modular scripts.

8.3.2. Badging System

While the badge system works fine, by relying on the save manager to store progress data and the scene manager to update itself with any newly-completed minigames, there are also other alternatives to make it work the same – conceptually. One method is to apply the ‘Don’t Destroy On Load’ code feature on the badge manager and have it traverse to the minigames and back, just like the scene manager. This way, the need to rely on both save and scene managers will be redundant, and the entire logic will be kept within the badge manager. However, the design choice was to actively reduce the main game’s footprint on the minigames as much as possible; mainly to ensure the freedom that the minigames had in their making, and due to the fact that the minigames had no standardized way of linking to the main game’s systems. Similarly, the badge manager could have also been made to depend solely on the save manager and ignore the scene manager. After all, while not a game object, the save manager

still existed and was accessible at all times. Nevertheless, the save manager is considered to be responsible for data storage, first and foremost, and adding game-affecting functionality to it seems counterintuitive and muddies its role.

8.3.3. State Saving & Loading System

Currently, and as explained above, the save manager is made to create an auto-save file before loading a minigame level, and then load this file after returning to the main game. This indirectly became a perk from the gameplay perspective; as players don't have the option to save their progress while playing a minigame, they may forget to save manually before starting one. If it happens that Methodyca is shut down for whatever reason – such as the browser window closing – the player can pick up from the moment right before starting the last minigame, saving them the trouble of instead having to retread their steps from their manual saves. Furthermore, as the rooms in the main game all act as checkpoints, the player's progress is again automatically saved after their return to the main game – again acting as back up for their playtime in case something shuts down Methodyca before they get a chance to save.

This process serves its purpose well, but is also somewhat inefficient. Looking closely, there shouldn't be a reason to load the save data after returning from a minigame, since the save manager is never removed nor closed – the latest data used in the auto-save are still stored in the save manager and don't need reloading and overwriting itself. The only reason this step is taken, however, is because the 'current scene' value in the save manager gets replaced with the minigame scene, and hence the game cannot load the main game until the data is restored from the file. As the save files are quite small in size, and doing this extra step has no observable detrimental impact on gameplay, it is not considered an issue. Conversely, if this step impairs gameplay flow and slows it down, due to larger data files for example, then the developer should consider optimizing this step. One possible solution is to make the save manager record two scene values: the current scene, for example the minigame the player is in; and the previous scene, which had the entry point to the current minigame level - typically somewhere in the main game. With the additional 'previous scene' value stored in the save manager and when the player wishes to return to the main game, there will be no more

requirement to force the save manager to reload the saved data only to know which scene the player should go to next - the save manager will just get this information from the 'previous scene' property.

8.3.4. User Feedback

During the playtesting sessions, a constantly observed issue was how the background music (BGM) of the minigames would carry over and 'stick' under the main game's own BGM. This again relates to the aforementioned problem of minigame developers not making the necessary considerations for their minigames to tie into the main game when they're done. At any rate, not all the blame should fall on them, as part of this responsibility also belongs to the main game's development team in coordinating and maintaining a sort of production pipeline for the minigame developers to adhere to – unfortunately, this was not available.

There was no actual feedback on this particular part of the game, and as can be seen in **Appendix B**, any feedback was mainly about the gameplay experiences of the minigames. This is understandable and expected, as the connection between the main game and the minigames is concealed from the player's senses, and players simply move between levels naturally without giving thought about such technicalities. More importantly, the lack of any feedback about something that shouldn't be noticeable in the first place means positive results: if players couldn't find or feel the presence of any problems while jumping from one level to the next, and if everything regarding this aspect of the game was going smoothly, then it's a sign of the system working as it should.

9. CONCLUSIONS

This chapter looks at the conclusions of the study, Methodyca as a game and its development, and general user feedback. The research questions posed at the beginning of this thesis are answered, and future prospects of this research study are laid out.

9.1. Methodyca as a Learning Game

Methodyca is now available for both students and the public to play; with 9 minigames that are also as standalone games in their own right, and 3 additional ones embedded within the full game. Methodyca was developed for the purpose to create a product that can help assist students with their learning of research methods via a game-based learning experience encapsulated in an adventure story-driven game. The game will be provided to the students of the Research Methods course at Tallinn University, to accompany their studies and help them play around with some of the theory material they work with in class.

From the development perspective, Methodyca has gone through a journey; from the very early stages of developing small individual pieces, to the realized systems and components that all work together to bring the holistic gameplay experience it has to offer. Those systems, at their current stage, have achieved a level of flexibility and modularity that should allow them to not only be used to expand Methodyca, if desired, but also be reused in similar game projects as well, where developers can either follow in the same footsteps of how Methodyca was created and fit it in different narrative and art style. It is not even necessary to take everything from Methodyca as a whole. Since the different systems were developed independently of one another and then made to 'talk' with each other, this means that a developer can simply pick and choose the one that suits their needs.

The self-reflection of the systems developed for Methodyca raised a few questions and possible alternatives to be considered, in addition to questioning the reasoning behind the design choices put into programming and building the game. Discussing and documenting how a project of this size was built (despite covering only the most critical parts and not the game in its entirety), as well as making its source code publicly available, will hopefully create

opportunities to build upon, and even critique this work, in both the academic and non-academic worlds.

9.2. Overall User Feedback

Although this research hoped to collect more results and feedback by employing two data collection methods, in the form of questionnaires and live play-testing sessions, the response rate for the online questionnaires – which amounted to 3 results only – was unfortunately too little to warrant looking into. Nevertheless, the turnaround for the live play-testing sessions was much more reassuring, with 13 participants playing the game and providing their feedback. The live sessions are even arguably more beneficial than the online questionnaires; where the questionnaires are done solely on the discretion of the participants, with no way for the developer to observe or validate their answers, the live sessions on the other hand allow the developer to observe subtle player actions, whom may either forget about or dismiss as unimportant to report. One such example was the attempt by some participants to drag items from the inventory, instead of clicking on them; which could have easily slipped by in the case of an online questionnaire that relies on players answering it after playing the game but on their own accord.

Based on the observations and participant feedback from the play-testing sessions, 7 unique game bugs were found that relate directly to the dialogue, interaction, and inventory systems. Of those 7 bugs, only a single bug, connected to the dialogue with one character, was able to completely break the game and force the participant to restart it again – luckily, the autosave feature helped to recover the progress until before the bug occurred, and the participants simply avoided the action that caused it and continued playing. This is quite an optimistic result, considering this was the very first play-testing session of Methodyca with participants that represent potential players of the game. Except from the 1 game-breaking bug, the other 6 that existed did not cause much harm to the gameplay experience, and players were able to normally play the game from start to finish – provided they avoid that one bug. This is not, by all means, an indication of near-perfect programming skills, which after over a year of developing such a sizable game by a small team would end up with 7 bugs across 5 different

game systems. It is an indication, however, of the usefulness and worthiness of consistent internal playtesting of the game within the entire team and also by the developers building the systems themselves. During the development of Methodyca, many bugs existed in the game, but many of them were also found, thanks to creating several prototypes of the game along the way which the team tested and reported their findings and feedback, similar to the play-testing sessions. The prototypes were tools that facilitated internal discussions over design and issues found in the game, and by the time the participants of this study's play-tests got to play the game, many of these issues were fixed. Additionally, that single game-breaking bug was easily found after the play-test sessions, and turned out to be an easy fix, which will be incorporated into the next version release of the Methodyca game. The remaining bugs will, of course, be rooted out and fixed as well in future releases.

Aside from game bugs, participants offered their feedback which touched on both design and development-related aspects, regarding the game systems discussed in this thesis and also about the game in general. The responses and suggestions, which can be viewed in full in **Appendix B**, were generally directed at players needing more hints and indications about how to find and use the inventory, as well as navigating and interacting with the game world in general. It is difficult to completely separate these suggestions into either design or development ones, as both go hand in hand. If a suggestion or recommendation is chosen to be acted upon, the game designers will be needed to make the choices of when, where, and how should an event like a hint occur in the game, while the game developers will need to figure out and decide how these events will go into the game files and code, in order to function as described by the designers.

Although the play-testing sessions focused on the bugs and issues that can be found in the game, the general feedback and participant reactions were both positive and supporting of the game, with some acknowledging the usefulness of such a game in a research methods course, from their own experience with studying it. The participants generally commented on the fun and varied artistic styles of the game, and how the various looks and play styles of the minigames made the game feel quite big and encouraging to explore all of them.

9.3. Revisiting the Research Questions

In this section, the 3 main research questions that were asked at the beginning of the thesis are recalled and answered.

9.3.1. What are the Common Elements in Point-&-Click Games?

While point-and-click games look and play differently from one another, and even cross over to other genres, like how The Longing mixed elements of the idle-game genre (STUDIO SEUFZ, 2020), they still share basic common elements that situate them in the point-and-click genre. Although these elements are independent of one another in isolation, they come together to form the holistic experience of the game. The elements were explored in this study, and are conceived in the form of a world navigation and interaction system, an inventory system, and a dialogue system. Puzzle-solving is another element that is found in all the point-and-click games investigated in this study, however it is not a system in itself, but a product that is born out of combining the world interaction, narrative, and inventory all together.

9.3.2. How to build a Modular & Expandable Point-&-Click Adventure Game?

In investigating the makings of Methodyca, this thesis brought forward the design strategies and techniques used to create a game whose systems may appear large and complex, but at the same time are flexible and modular. Locations can be imagined as rooms in 3D space, placed together to mimic a natural pathing sequence and flow, and hence many rooms can be put together to create larger, more open worlds and levels. The inventory system can be broken down into smaller parts, where the inventory manager decides on the limit of items the player can carry and lists them, while the inventory slots can be setup in any way the developer desires, while still having the choice to freely set the number of items a player can carry as seen fit with the game. The badges system, while optional, can be utilized in a similar fashion to the inventory system. The dialogue and narrative system ended up using Inky – a 3rd party plugin – to support it, and the nature of the plugin itself makes it flexible and modular, to be used and reused as many times as the developer needs it. Its flexibility even went as far as streamlining the narrative building process between the developers and narrative designers. Finally, a state

saving and loading system lets everything come together to make the entire game experience persistent and smooth, allowing not only the player to leave the game and pick it back up at will, but also the other systems to function properly across levels and minigames. All these systems combined build a point-and-click game experience, and they all empower both designer and developer to resize the game at will.

9.3.3. What are the Lessons Learned from Player Feedback & this Retrospective Study?

This study examined some alternatives and drawbacks to the systems that were developed for Methodyca. The biggest lesson learned from this self-reflection is the need to plan games that have multiple systems in them; as with each new system added the dependencies increase and more complexity levels become layered onto each other. Methodyca was perhaps taken a bit lightly at the start of development, and the amount of work needed to complete it was underestimated. A dialogue system was planned to be developed in-house, only to be found out later on how large of a project this can be on its own, thus leading to the decision to move to a 3rd party plugin. The save system was designed and implemented in the second half of development, which introduced several issues and challenges, mainly in coordinating the saving and loading of data among systems that depended on each other to function properly. These two hurdles would have most likely been overcome if the systems were all pre-planned, and a hierarchy of dependencies was drawn up, to make not only the development of the systems more efficient, but to also ease the integration step between them. The study also looked at and discussed other lessons that were learned, but this one could arguably be the most important of them all.

Looking at the external user feedback from play-testing, the first one may be somewhat obvious: No matter how well a developer may believe their system is working and it being bug-free, there will always be bugs. Simply put, the developer is only one person, and it is usually difficult for one person to break the habit of how they play a game and test it. Therefore, the developer may catch plenty of bugs on their own, but they tend to end up becoming numb to their own work, and repeating the same way of testing, or even skipping parts of the game if

they believe in their mind that these parts work fine. Having others test the work with a fresh set of eyes helps in looking at things from a different perspective. It doesn't necessarily have to be bugs that surface, but also player habits that the developer isn't aware of can show themselves. These insights provide a great deal of knowledge to both developer and designer to revise their work and previous assumptions. The second lesson learned is regarding the assumption made about the participants of the live sessions. At first, it was assumed that participants who rarely played video games or even not at all would not provide useful input or feedback on the game. The feedback data collected from the play-testing sessions proved that this was a wrongful assumption, as participants who identified themselves as non-gamers were able to provide as much a useful input as the others. Lastly, from the bug-reporting results of the play-testing sessions, the benefits of internal play-testing evidently show themselves. Play-testing should not just be an activity that takes place at the end of the game development project, but is an ongoing process that goes along with development, to continually make sure that problems can be tackled as soon as they crop up.

9.4. Changing the Rules of the Game

Methodyca is a game that manages to combine the tropes of the point-and-click genre with learning material that is targeted to higher education students. And it also managed to bring together a wide variety of gameplay mechanics and art styles into a single experience, which is quite unorthodox for a single game to have such diversity. To be able to make this all come true, from the development aspect, may be considered a nightmare to most developers. Nevertheless, it ended up becoming a playable reality, with the potential to add even more minigames to it. The efforts put so far, in making Methodyca, the publishing of this study, and releasing the game code to the public, attempt to contribute to the academic body in the scope of game research and development, as well as students and professionals in the game industry - not to mention the learning goals of Methodyca as a game itself.

9.5. Limitations & Further Research

While this study hoped to cover the most paramount aspects of the development of Methodyca, there still exists parts of it that should be open for further discussions. Methodyca

includes a mechanic that allows the player to switch between 2 dimensions, and play between them simultaneously. The development of such mechanics was not included in this study, considering the large work that went into realizing it. For now, the only possible way to study it is by reviewing the game's source code made available on the online repository. In addition, in RtD, there is one more step that comes after this thesis, as explained by Zimmerman & Forlizzi (2014), which is to repeat. To reach the fruition point of the systems developed in Methodyca through the RtD methodology, the user feedback and self-reflection obtained from this study should be taken forth into new projects by fellow scholars and developers, and re-applied in other contexts, then further refined.

The systems that were studied and discussed in this thesis were created to satisfy the needs of one game in one genre of games. Although this study and reflection argue that the systems developed are flexible, modular, and can be reused, this statement needs to be put to the test in future projects, and the systems be criticized and refined. One potential possibility is to learn whether the concepts applied in these systems apply in other game engines besides the Unity engine, or in even other game genres. Furthermore, the play-testing carried out in Methodyca did not uncover enough information about the badge or save system, due to the short play time. These systems will need to be tested in the longer term to verify their functionality. Bugs were found, and feedback was provided, which warrants that the systems questioned and analyzed in this study will require revisiting, and perhaps even redesigning some of their parts. This introduces the need for further research on the state of Methodyca and its systems after several releases and revisions.

Methodyca was developed as a browser game to be played only on computer devices, with no support for mobile devices. In today's world where mobile devices are ubiquitous, the lack of a mobile version of Methodyca can be considered a liability to it, and a call to action. This invites further research on how Methodyca can be ported to mobile devices, and how does that affect the systems developed for it?

Methodyca started out on the basis of how other games of the genre should be made to look like, and then took those rules and turned them around, by mixing and matching elements from

card games, visual novels, puzzle games, and more. It also went a step further, by incorporating game mechanics like playing in two worlds simultaneously and switching between them at will. Methodyca may not be considered at the same level of million-dollar game projects, but it still attempted to create interesting twists in the point-and-click genre and game-based learning. This at least opens the door for new further studies and discussion about game design, game development, and game-based learning.

REFERENCES

5 things you need to know about playtesting. (2019, June 7). Massive. https://www.massive.se/blog/consumer-experience/5-things-you-need-to-know-about-playtesting/

Adams, E. (2010). *Fundamentals of game design* (2nd ed). New Riders.

Bartle, R. A. (2004). *Designing Virtual Worlds*. New Riders.

Beck, K. (2003). *Test-driven Development: By Example*. Addison-Wesley Professional.

Berg Marklund, B., Engström, H., Hellkvist, M., & Backlund, P. (2019). What Empirically Based Research Tells Us About Game Development. *The Computer Games Journal*, *8*(3–4), 179–198. https://doi.org/10.1007/s40869-019-00085-1

Blake, V., & Gerblick, J. (2021, January). *Cyberpunk 2077 corrupted save bug fixed in new patch*. Gamesradar. https://www.gamesradar.com/your-cyberpunk-2077-save-may-be-corrupted-if-you-pick-up-too-many-items/

Borg, M., Garousi, V., Mahmoud, A., Olsson, T., & Stalberg, O. (2020). Video Game Development in a Rush: A Survey of the Global Game Jam Participants. *IEEE Transactions on Games*, *12*(3), 246–259. https://doi.org/10.1109/TG.2019.2910248

Brackeys. (2017, August 16). *INVENTORY UI - Making an RPG in Unity (E05)*. https://www.youtube.com/watch?v=w6_fetj9PIw

CD Projekt. (2020). *Cyberpunk 2077* [PC]. CD Projekt Red.

Choi, J. O., Forlizzi, J., Christel, M., Moeller, R., Bates, M., & Hammer, J. (2016). Playtesting with a Purpose. *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, 254–265. https://doi.org/10.1145/2967934.2968103

Deponia [PC]. (2012). Daedalic Entertainment GmbH.

Ken Follett's The Pillars of the Earth [PC]. (2017). Daedalic Entertainment GmbH.

Epic Games. (2008). *Gears of War 2* [Xbox]. Microsoft Game Studios.

Fine, R. (2017, August 11). *UnityScript's long ride off into the sunset—Unity Technologies Blog*. https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

Frayling, C. & Royal College of Art. (1993). *Research in art and design*. Royal College of Art.

Freedman, E. (2018). Engineering Queerness in the Game Development Pipeline. *Game Studies*, *18*(3). http://gamestudies.org/1803/articles/ericfreedman

FromSoftware, Inc. (2011). *Dark Souls* [PC]. Namco Bandai Games

G2A.COM. (2020, October 1). *15 Most Popular Point and Click Adventure Games [2020]*. G2A News. https://www.g2a.com/news/features/15-most-popular-point-and-click-adventure-games/

Hamari, J., & Eranti, V. (2011). *Framework for Designing and Evaluating Game Achievements* (Vol. 115).

Homeworlds. (n.d.). Spyro Wiki. Retrieved April 29, 2021, from https://spyro.fandom.com/wiki/Homeworlds

ICOM Simulations. (1985). *Déjà Vu* [PC]. Mindscape.

Imagine Media. (1996). *NEXT Generation Issue #15 March 1996*. http://archive.org/details/nextgen-issue-015

Sorcery! [Mobile]. (2013). Inkle.

80 Days [PC]. (2014). Inkle.

Heaven's Vault [PC]. (2019). Inkle.

Insomniac Games. (1998). *Spyro the Dragon* [PlayStation]. Sony Computer Entertainment.

Jakobsson, M. (2009). *The Achievement Machine: Understanding the Xbox Live Metagame*.

Kortmann, R., & Harteveld, C. (2009). *Agile game development: Lessons learned from software engineering*.

Kyratzes, J. (2016). *The Sea Will Claim Everything* [PC]. Jonas Kyratzes.

Lucasarts. (1990). *The Secret of Monkey Island* [PC]. Lucasfilm Games.

Lucasarts. (2009). *The Secret of Monkey Island: Special Edition* [PC]. Lucasfilm Games.

Maps—Dark Souls. (2019, August). Dark Souls Wiki. https://darksouls.wiki.fextralife.com/maps

Marchiori, E. J., Torrente, J., del Blanco, Á., Moreno-Ger, P., Sancho, P., & Fernández-Manjón, B. (2012). A narrative metaphor to facilitate educational game authoring. *Computers & Education*, *58*(1), 590–599. https://doi.org/10.1016/j.compedu.2011.09.017

Martin, P. (2018). The Intellectual Structure of Game Research. *Game Studies*, *18*(1). http://gamestudies.org/1801/articles/paul_martin

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education.

Syberia [PC]. (2002). Microïds.

Moss, R. (2011, January 27). *A truly graphic adventure: The 25-year rise and fall of a beloved genre*. Ars Technica. https://arstechnica.com/gaming/reviews/2011/01/history-of-graphic-adventures.ars

Osborne O'Hagan, A., Coleman, G., & O'Connor, R. (2014). Software Development Processes for Games: A Systematic Literature Review. In *Communications in Computer and Information Science* (Vol. 425). https://doi.org/10.1007/978-3-662-43896-1_16

PC Savegames. (2018). *The Secret of Monkey Island: Special Edition Savegames*. PC Savegames. http://www.nicouzouf.com/en/?id=thesecretofmonkeyislandspecialedition

RainingChain, Demajen, AlexanderXIII, Zigmatism, Seanpr, Mickley, fractalsonfire, Phaolo, Enthi, iloveNCIS7, Romacus, omarcinimancini, FurudoErika, foxyfille, Axew, Sans, & swami origami. (n.d.). *Hollow Knight Interactive Map—Godmaster*. Retrieved April 29, 2021, from https://scripterswar.com/hollowknight/map

Revolution Software. (1996). *Broken Sword* [PC]. Virgin Interactive Entertainment.

Ribbing, V., & Melander, L. (2016). *Examining the Souls' Series Level Design*.

Robbs, M. (n.d.). How Did The Original NES Save Games? (Did It?). *Retro Only*. Retrieved April 28, 2021, from https://retroonly.com/how-did-the-original-nes-save-games/

Roggema, R. (2016). Research by Design: Proposition for a Methodological Approach. *Urban Science*, *1*(1), 2. https://doi.org/10.3390/urbansci1010002

Santa Monica Studio. (2007). *God of War II* [PlayStation 2]. Sony Computer Entertainment.

Shin, Y., Kim, J., Jin, K., & Kim, Y. B. (2020). Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning. *IEEE Access*, *8*, 51593–51600. https://doi.org/10.1109/ACCESS.2020.2980380

Sirlin, D. (2008). *Saving the Day: Save Systems in Games*. https://www.gamasutra.com/view/feature/130352/saving_the_day_save_systems_in_.php

The Journey Down. (2010). Skygoblin.

STUDIO SEUFZ. (2020). *The Longing* [PC]. Application Systems Heidelberg.

Hollow Knight [PC]. (2017). Team Cherry.

Uccello, A. (2017, September 1). *How to Save and Load a Game in Unity*. Raywenderlich.Com. https://www.raywenderlich.com/418-how-to-save-and-load-a-game-in-unity

Unity Forum. (2009). *Control the GameObject Start() order*. Unity Forum. https://forum.unity.com/threads/control-the-gameobject-start-order.19976/

Unity Technologies. (n.d.-a). *Get Unity—Download Archive*. Unity. Retrieved May 4, 2021, from https://unity3d.com/get-unity/download/archive

Unity Technologies. (n.d.-b). *Unity QA - LTS Releases*. Unity. Retrieved May 4, 2021, from https://unity3d.com/unity/qa/lts-releases

Zichermann, G., & Cunningham, C. (2011). *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps* (1st edition). O'Reilly Media.

Zimmerman, J., & Forlizzi, J. (2014). Research Through Design in HCI. In J. S. Olson & W. A. Kellogg (Eds.), *Ways of Knowing in HCI* (pp. 167–189). Springer New York. https://doi.org/10.1007/978-1-4939-0378-8_8

Zimmerman, J., Stolterman, E., & Forlizzi, J. (2010). *An Analysis and Critique of Research through Design: Towards a formalization of a research approach*. 10.

APPENDIX A: ONLINE QUESTIONNAIRE

The following is the questionnaire as it was made available online. It should be noted that the web links provided may not be accessible any longer.

Methodyca Play-Test Feedback

Thank you for taking the time to fill-in this feedback form for the playtesting of Methodyca game.
This goal of this questionnaire is for you to report your experience on playing Methodyca, and for the development team to understand if the game is working properly or not. The information here will also be used to write a research document about the development of Methodyca.
In the following section, you will be asked for some information about yourself. This information will not be used in any personal manner, and none of it can lead to you. Information like your name, email, or contact information will NOT be collected nor are they required for this questionnaire. The information requested here is mainly for statistical and categorisation purposes.
To carry out this questionnaire, you will need to first play Methodyca using the link provided to you, or you can use the link below. This questionnaire is interested in the game's functionality more than design. If you feel stuck at any point during the game and are not able to proceed, please consult the quick guide document linked below.
The mini-games can be skipped (for testing purposes) using the special buttons with the text "Complete Game" in each of the mini-games. If you wish, you can still complete the mini-games the normal way instead of skipping them. However, this questionnaire will not ask you about them, so it's up to you.
The best way to answer this questionnaire is to provide your feedback to the questions as you play each level, instead of waiting until completing the entire game first.
For the most optimal experience, please use Google Chrome, Mozilla Firefox, or Microsoft Edge (if you're on a Windows PC). Methodyca is not supported on the Apple Safari browser, and may break or not start at all.
N.B.: The game is meant to be played on a computer, and will not perform properly on a mobile device.
If you have any questions, please write an email to shenawy@tlu.ee
Link to Methodyca (test version): http://www.tlu.ee/~shenawy/RM%20Game/210415%20-%20Methodyca%20v1.0.3_PT/
Link to Guide:
https://docs.google.com/document/d/18JKVjW0k7WiA4xApqUuQeOxP4Xf5QSVStV9DSgBe0LE/edit?usp=sharing

Information About Yourself

What is your age group?

- ☐ Under 20
- ☐ 20 - 30
- ☐ 30 - 40
- ☐ Over 40

What is your education level?

- ☐ Higher Education - University student or graduate
- ☐ Secondary education - High school
- ☐ Below Secondary education

What is your English language level?

- ☐ Native
- ☐ Fluent
- ☐ Intermediate
- ☐ Basic

How familiar are you with research methods?

- ☐ I know and actively use them in my work and/or studies
- ☐ I know of them, but don't actively use them in my work or studies
- ☐ I have no knowledge about research methods

How often do you play video games?

- ☐ Regularly - several times a week
- ☐ Occasionally - a few times a month
- ☐ Rarely or not at all

First Level (Act 1)

Were you able to interact with world objects? (e.g., door, safe, desk, portal, etc.)

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to pick up/ collect items (e.g., key, crystals)?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to start the desk sorting mini-game?

- ☐ Yes
- ☐ No
- ☐ Other: _____

After winning the desk sorting mini-game, were the crystals at the desk charged and can be picked up?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to use the portal to reach the next level (Act 2)?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Did you see a video cut-scene play after you used the activated portal?

- ☐ Yes
- ☐ No

If there are any issues or bugs you faced while playing the game until this point, please mention them here.

_____

Second Level (Act 2)

Were you able to interact with world objects? (e.g., gates, characters, etc.)

- ☐ Yes
- ☐ No
- ☐ Other: _____

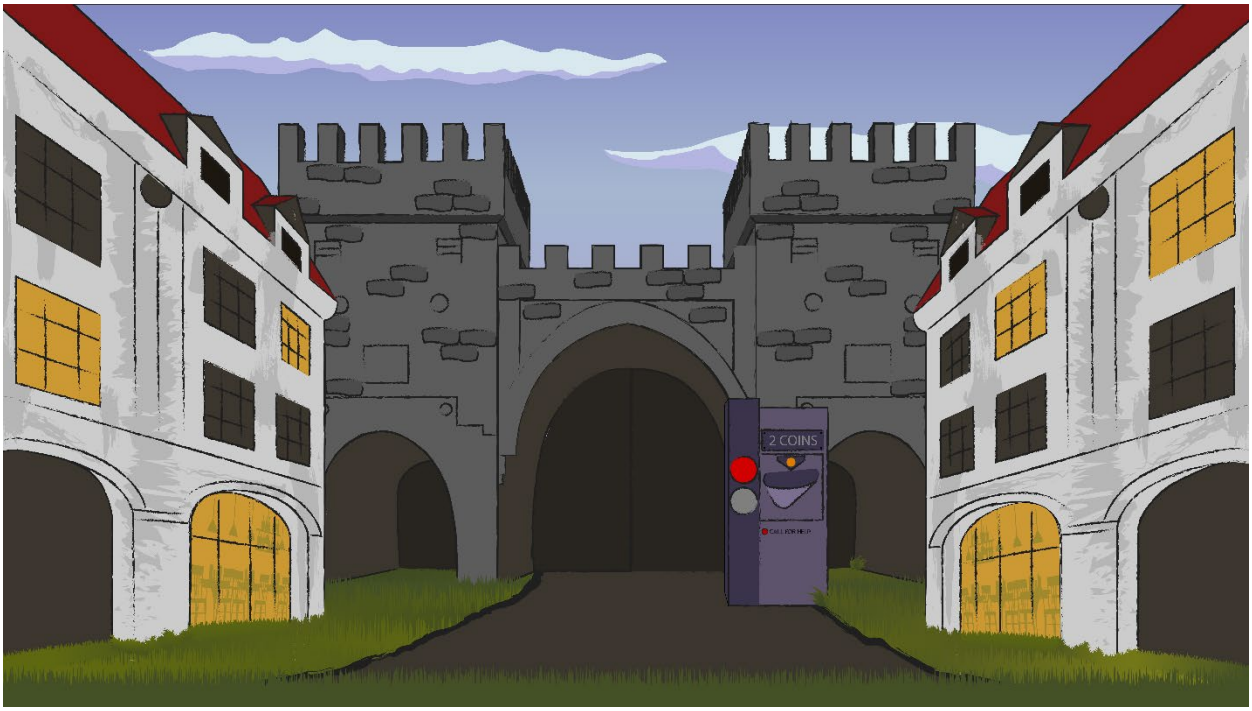
Were you able to talk to the in-game characters and choose dialogue options?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to travel between dimensions using the dimension-switching device in your inventory?

- ☐ Yes
- ☐ No

First Area (artwork by Elena Gorshkova)



Were you able to start the mini-games in the first area?

- ☐ Yes
- ☐ No

If the answer to the above is No, which mini-game(s) did not start?

- ☐ Game at cafe, left of toll gate (Questioniser)
- ☐ Game at binoculars tripod, right of toll gate (Binoculars)
- ☐ Game at big house door, right of toll gate (Interview Simulator)
- ☐ Game at library, right of toll gate (Jammr)

Which mini-games did you play in the first area?

- ☐ Questioniser
- ☐ Binoculars
- ☐ Interview Simulator
- ☐ Jammr

Second Area (artwork by Iryna Selina)

Were you able to start the mini-games (dialogues about methodologies) in the second area?

- ☐ Yes
- ☐ No

If the answer to the above is No, which mini-game(s) did not start?

- ☐ Game at brown house, left of bridge (Survey)
- ☐ Game at IT company, left of bridge (Design Research)
- ☐ Game at newspaper building, left of bridge (Case Study)
- ☐ Game at green house, right of bridge (Experiment)

Which methodology-related dialogues did you play in the second area?

- ☐ Survey
- ☐ Design Research
- ☐ Case Study
- ☐ Experiment

If you faced any problems or bugs in any of the methodology-related dialogues, please state them here.

_____

Third Area (artwork by Anni Hõ)

Were you able to start the mini-games in the third area?

- ☐ Yes
- ☐ No

If the answer to the above is No, which mini-game(s) did not start?

- ☐ Game at Tardis, right of mansion & hill (Participatory Loop)
- ☐ Game at dog kennel, right of mansion & hill (ProtoEscape)
- ☐ Game in glass house, left of mansion & hill (Persona Game)

Which mini-games did you play in the third area?

- ☐ Participatory Loop
- ☐ ProtoEscape
- ☐ Persona Game

Did you receive the reward items in the first and third areas (coins and punch cards) to progress in the game?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to travel back and forth between the different areas (after you unlocked them)?

- ☐ Yes
- ☐ No
- ☐ Other: _____

If there are any issues or bugs you faced while playing the game until this point, please mention them here.

_____

Third Level (Act 3)

Were you able to talk to the supervisor character and choose dialogue options?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to start the research papers mini-game at the computer device?

- ☐ Yes
- ☐ No
- ☐ Other: _____

After winning the Research Papers Please mini-game, were the crystals at the computer device charged and can be picked up?

- ☐ Yes
- ☐ No
- ☐ Other: _____

Were you able to use the portal to complete the level?

- ☐ Yes
- ☐ No
- ☐ Other: _____

If there are any issues or bugs you faced while playing the game until this point, please mention them here.

_____

End Game

Does the 'Submit A Report' button work and open a new browser window?

- ☐ Yes
- ☐ No
- ☐ It opens a website link but in the same page of the game (the game is gone)
- ☐ Other: _____

In the previous question, does the link open a page where you can submit a research plan application?

- ☐ Yes
- ☐ No
- ☐ A page opens but it's not for submitting an application / empty
- ☐ Other: _____

Overall Gameplay Questions

Were you able to access your inventory and see the picked up items in it?

- ☐ Yes
- ☐ No

Were you able to use the items in your inventory (e.g., key, crystals, coins, etc.) with world objects?

- ☐ Yes
- ☐ No

If you had any problems or bugs with the inventory and using items in it, please mention them here.

_____

Have you tried saving/loading your game using the 'Save Game' or 'Load Game' apps in your phone in the inventory?

- ☐ Yes
- ☐ No

If you had any problems or bugs with saving and/or loading your save games, please mention them here.

_____

Did you receive badges for all the mini-games you won in the 'Badges' app in your mobile phone inventory item?

- ☐ Yes
- ☐ No

If you have successfully completed any of the mini-games and did not receive a badge for it, please check it here.

- ☐ Desk Sorting mini-game (Data Charger)
- ☐ Questioniser
- ☐ Binoculars
- ☐ Interview Simulator
- ☐ Jammr
- ☐ Survey (methodology dialogue mini-game)
- ☐ Design Research (methodology dialogue mini-game)
- ☐ Case Study (methodology dialogue mini-game)
- ☐ Experiment (methodology dialogue mini-game)
- ☐ Participatory Loop
- ☐ ProtoEscape
- ☐ Persona Game
- ☐ Research Papers Please

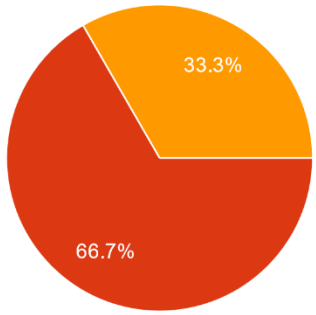
If there's anything else you wish to add or mention, please write it here.

_____

Following is the summary of feedback received from 3 (three) questionnaire responses at the time of this thesis' publication.

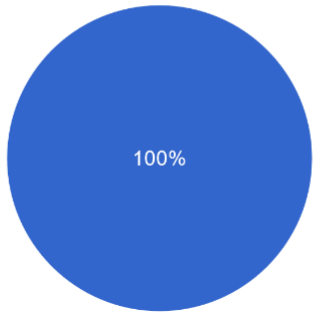
What is you age group?

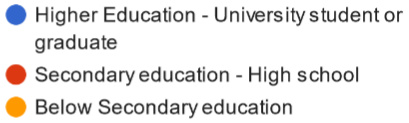
3 responses



What is your education level?

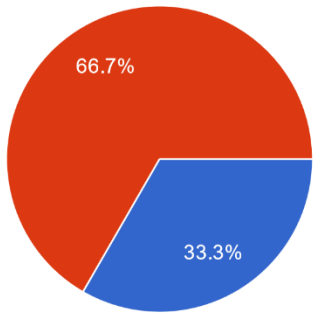
3 responses





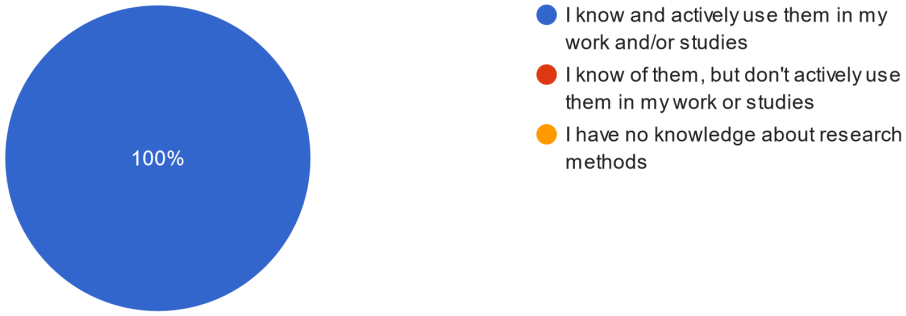
What is your English language level?

3 responses



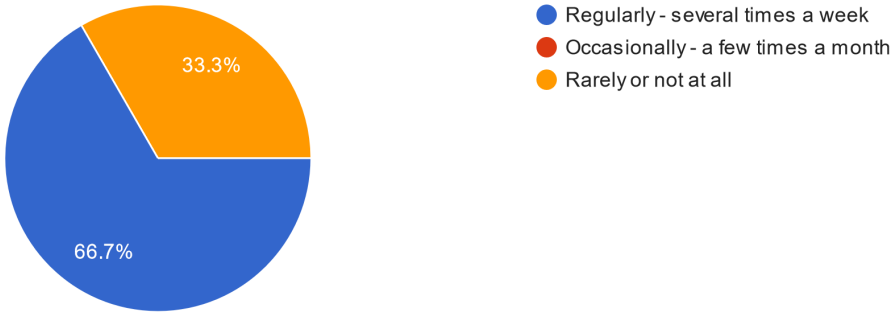
How familiar are you with research methods?

3 responses



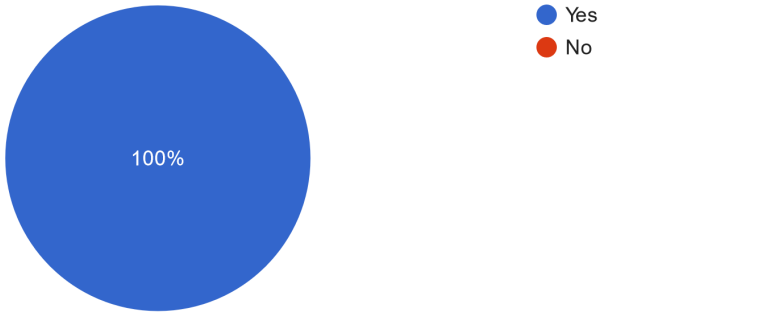
How often do you play video games?

3 responses



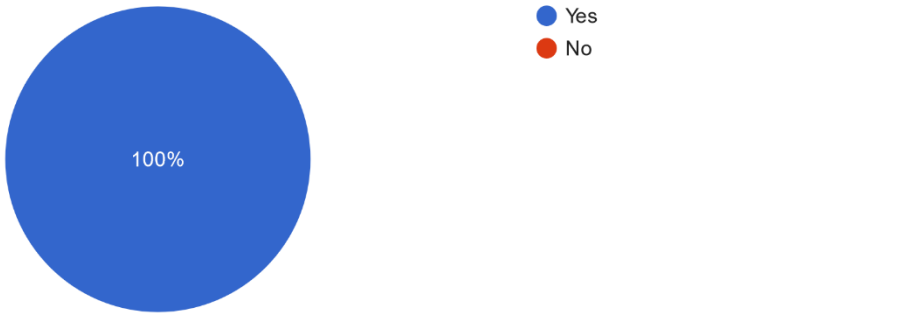
Were you able to interact with world objects? (e.g. door, safe, desk, portal, etc.)

3 responses



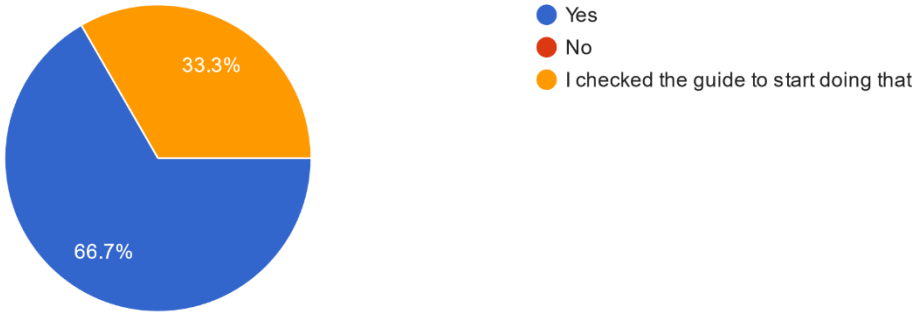
Were you able to pick up/ collect items (e.g. key, crystals)?

3 responses



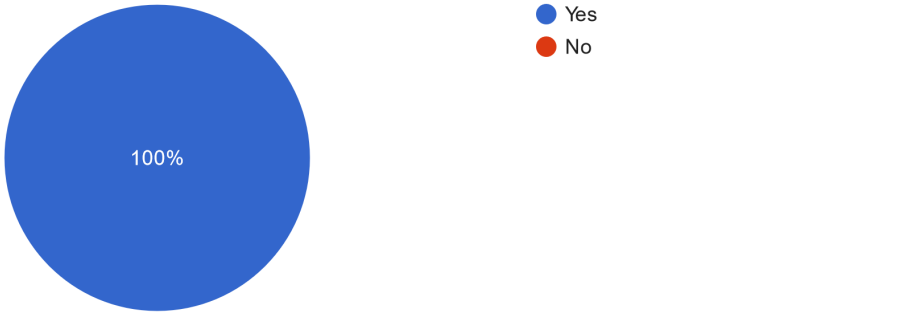
Were you able to start the desk sorting mini-game?

3 responses



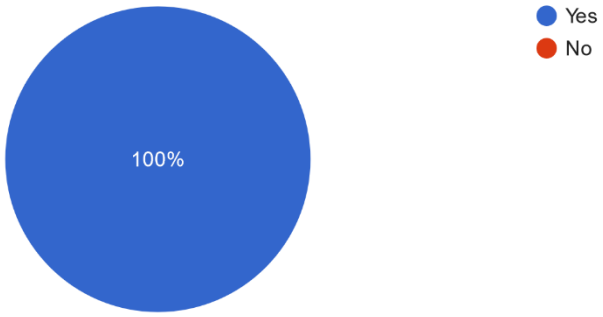
After winning the desk sorting mini-game, were the crystals at the desk charged and can be picked up?

3 responses



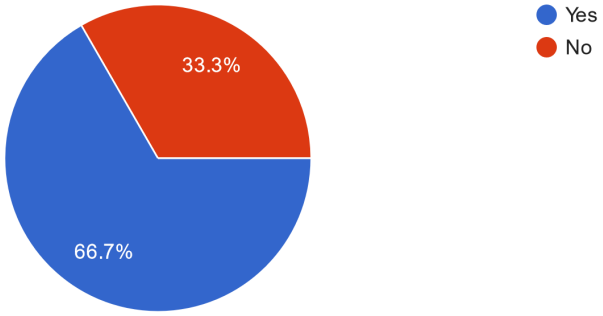
Were you able to use the portal to reach the next level (Act 2)?

3 responses



Did you see a video cut-scene play after you used the activated portal?

3 responses

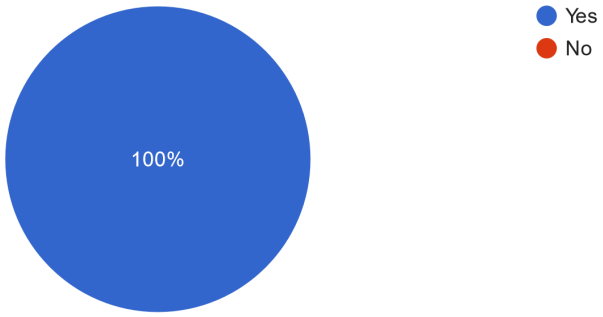


If there are any issues or bugs you faced while playing the game until this point, please mention them here.

1. The text in the dialogue was hard to see on the game background.
2. top bar with collected objects is hidden and it is no visual hint that something is there
3. The cut scene after placing the crystals didn't activate immediately

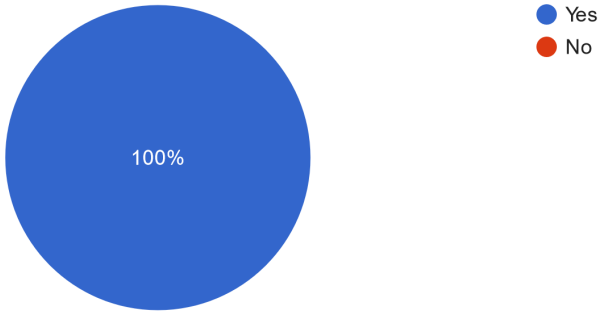
Were you able to interact with world objects? (e.g. gates, characters, etc.)

3 responses



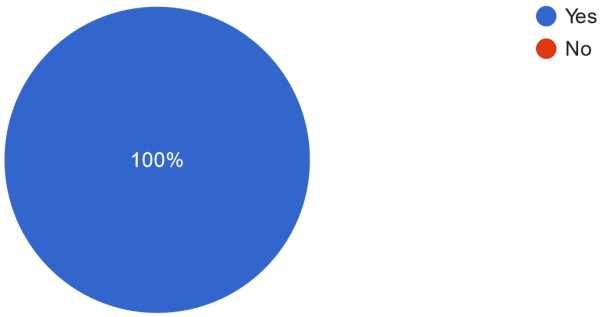
Were you able to talk to the in-game characters and choose dialogue options?

3 responses



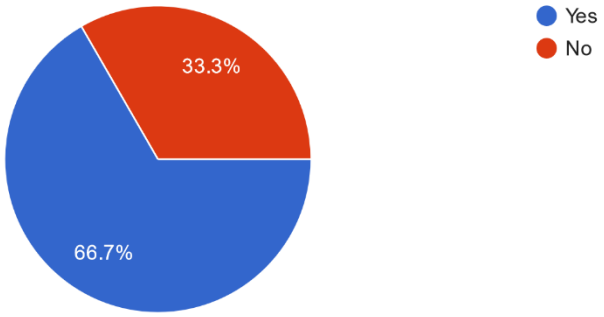
Were you able to travel between dimensions using the dimension-switching device in your inventory?

3 responses



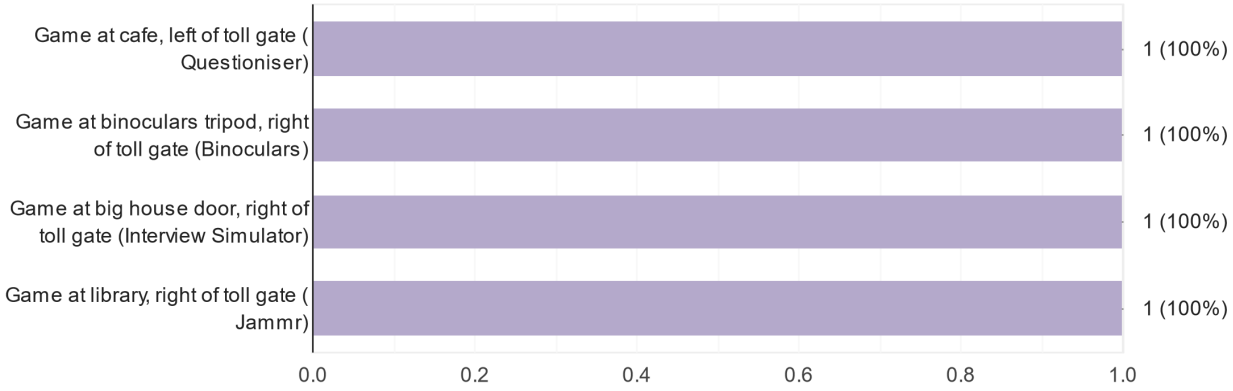
Were you able to start the mini-games in the first area?

3 responses



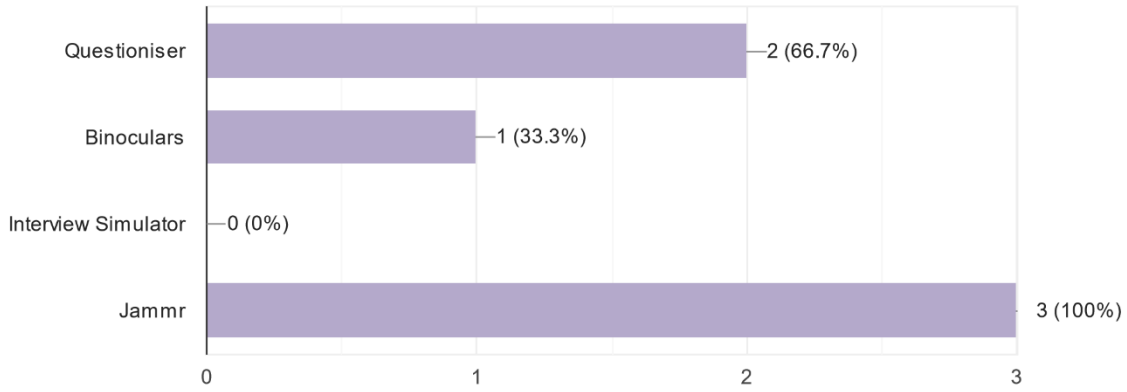
If the answer to the above is No, which mini-game(s) did not start?

1 response



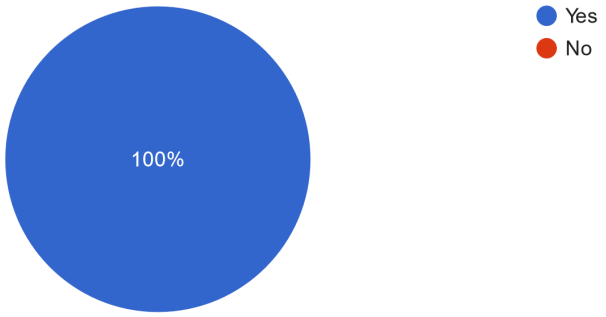
Which mini-games did you play in the first area?

3 responses



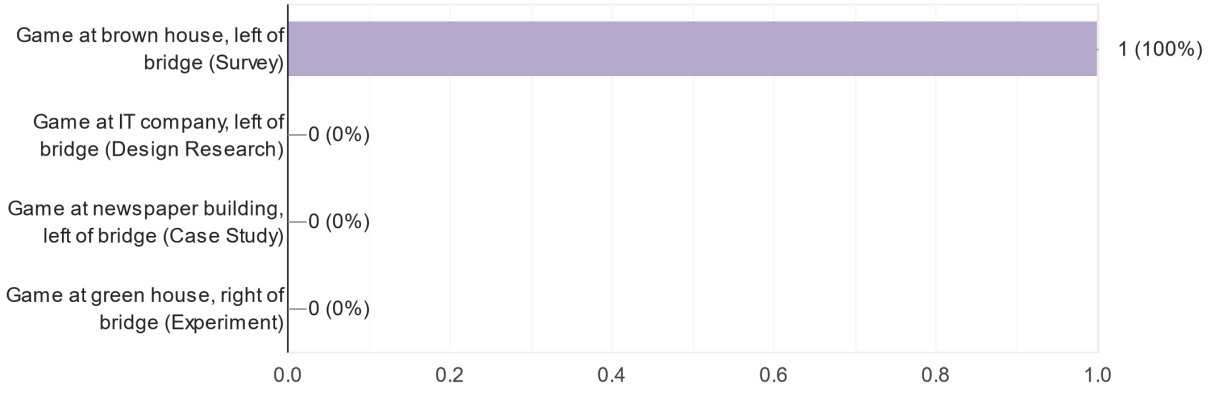
Were you able to start the mini-games (dialogues about methodologies) in the second area?

3 responses



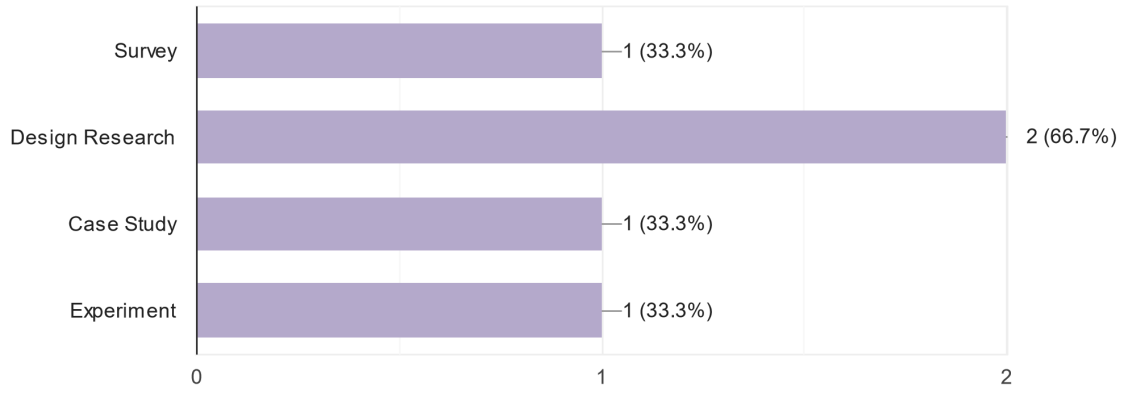
If the answer to the above is No, which mini-game(s) did not start?

1 response



Which methodology-related dialogues did you play in the second area?

3 responses

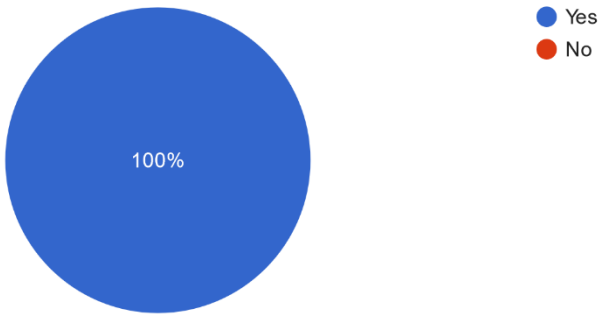


If you faced any problems or bugs in any of the methodology-related dialogues, please state them here.

1. I was able to trigger dialogues in places of minigames.

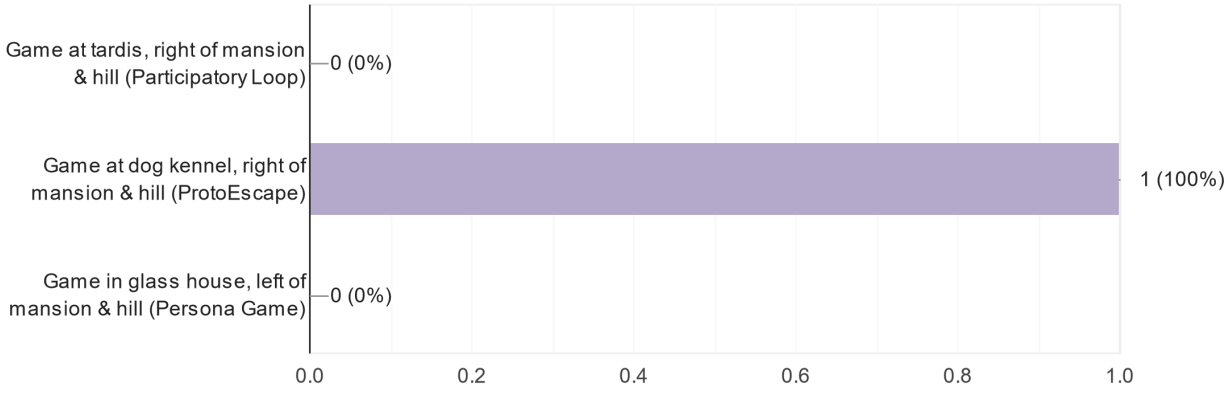
Were you able to start the mini-games in the third area?

3 responses



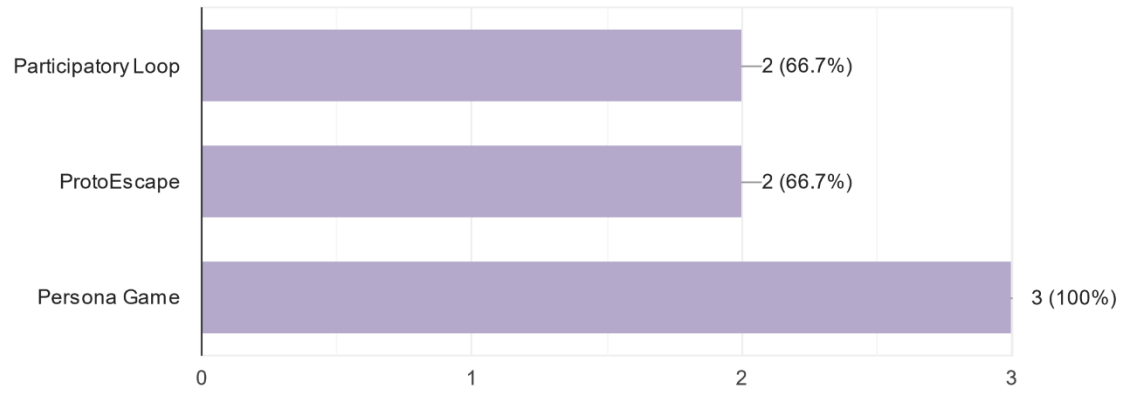
If the answer to the above is No, which mini-game(s) did not start?

1 response



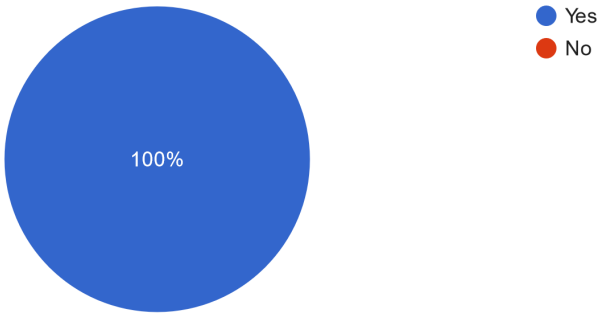
Which mini-games did you play in the third area?

3 responses



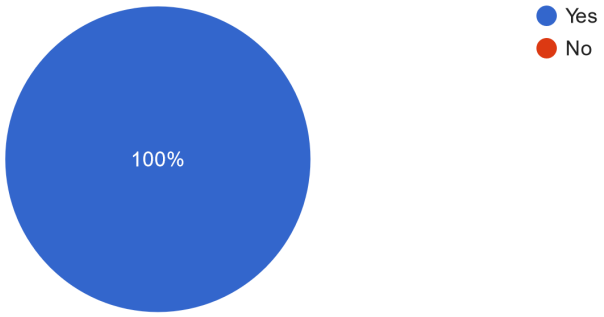
Did you receive the reward items in the first and third areas (coins and punch cards) to progress in the game?

3 responses



Were you able to travel back and forth between the different areas (after you unlocked them)?

3 responses

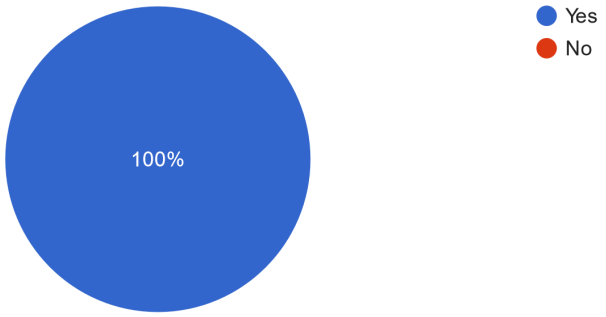


If there are any issues or bugs you faced while playing the game until this point, please mention them here.

1. No menu in participatory loop game.
2. After talk with mapcreater I stuck (Area First)... clicking everywhere but did not find a way to go. I refresh the page and continue without talking with this character
3. None

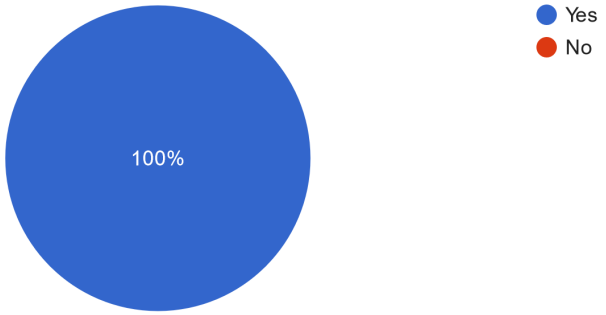
Were you able to talk to the supervisor character and choose dialogue options?

3 responses



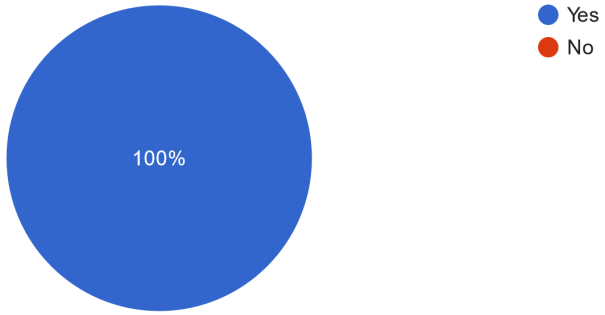
Were you able to start the research papers mini-game at the computer device?

3 responses



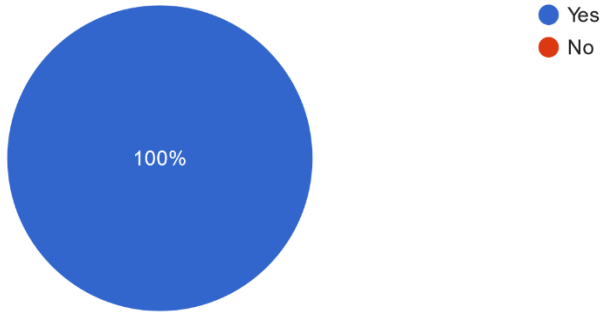
After winning the Research Papers Please mini-game, were the crystals at the computer device charged and can be picked up?

3 responses



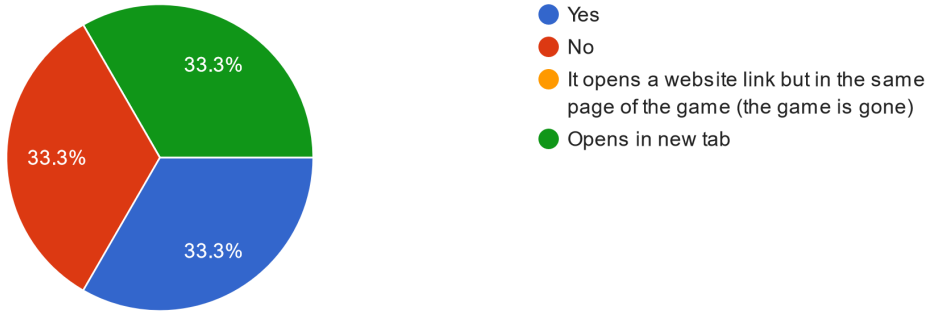
Were you able to use the portal to complete the level?

3 responses



Does the 'Submit A Report' button work and open a new browser window?

3 responses

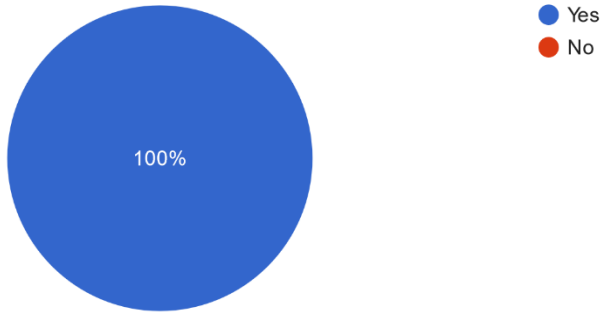


In the previous question, does the link open a page where you can submit a research plan application?

3 responses

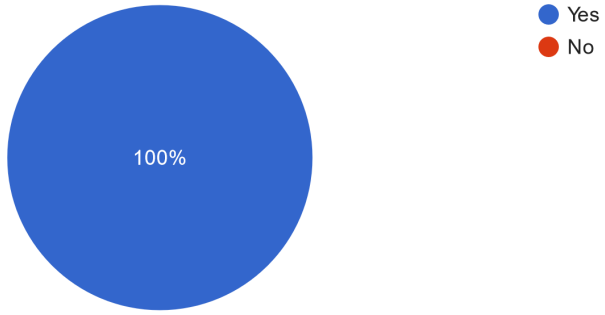
Were you able to access your inventory and see the picked up items in it?

3 responses



Were you able to use the items in your inventory (e.g. key, crystals, coins, etc.) with world objects?

3 responses

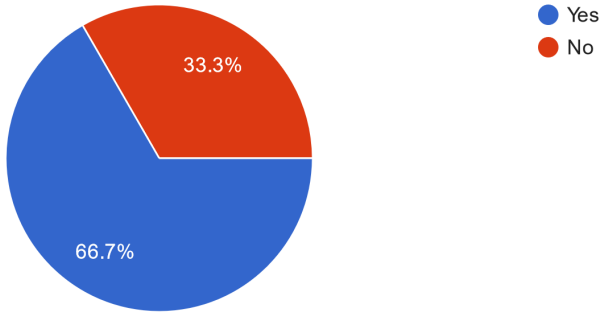


If you had any problems or bugs with the inventory and using items in it, please mention them here.

1. hint for inventory tab

Have you tried saving/loading your game using the 'Save Game' or 'Load Game' apps in your phone in the inventory?

3 responses

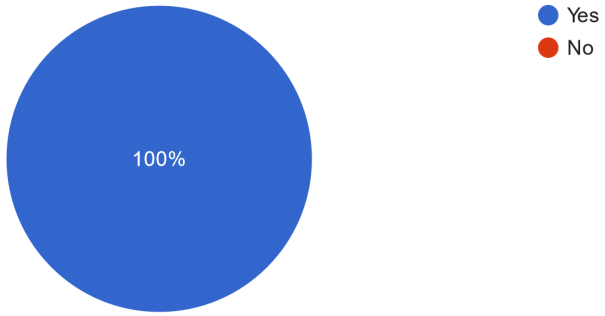


If you had any problems or bugs with saving and/or loading your save games, please mention them here.

1. I did not check badges during the game, so next question is not relevant for me.
2. It doesn't load in the coins

Did you receive badges for all the mini-games you won in the 'Badges' app in your mobile phone inventory item?

3 responses



APPENDIX B: LIVE PLAY-TESTING RESULTS

The following table lists all the observations and feedback collected from the 13 individuals who participated in the live play-testing sessions of Methodyca.

The 'Type' column is used to group the observations and feedback by systems type, as follows:

- **INT**: Interaction and Navigation;
- **INV**: Inventory system;
- **DLG**: Dialogue system;
- **BDG**: Badging system;
- **SAV**: State saving and loading system
- **MGM**: Minigames-related; and
- **GEN**: General observations and feedback.

No.	Subject Information	Observations	Type	Feedback	Type
1	**Age Group** 31 - 40 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** None **Time Spent Playing Video Games** Occasionally - A few times a month	- BUG: Map Maker dialogue breaks at first meeting. At end of dialogue player cannot interact with anything and game needed to be reset and continue from last save.	**DLG**	- Data Charger minigame: Need indication for charging levels of crystals.	**MGM**
		- BUG: The same research methodology minigame can be played in both dimension after one is already won. Noted with design-related research story.	**MGM**		
2	**Age Group** 31 - 40 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent	- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Suggests allowing to click and drag items from inventory.	**INV**
				- Game should provide tips on how to find the inventory.	**INV**

No.	Subject Information	Observations	Type	Feedback	Type
	Familiarity with Research Methods None **Time Spent Playing Video Games** Rarely or not at all				
3	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows and actively uses them in work and/or studies **Time Spent Playing Video Games** Regularly - Several times a week	- BUG: Cabinet in supervisor's office. Message saying can't open cabinet comes up after clicking and opening it.	**INT**	- Directions on how to play the game aren't clear enough.	**GEN**
		- BUG: Toll machine. Message saying can't use the machine comes up after giving 2 coins and using it as intended.	**INT**		
		- BUG: Dimension switching device in Act 3 is changing graphic as if it's working when clicked.	**INV**		

No.	Subject Information	Observations	Type	Feedback	Type
4	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows of them, but doesn't actively use them in work nor studies **Time Spent Playing Video Games** Regularly - Several times a week	- BUG: Cabinet in supervisor's office. Message saying can't open cabinet comes up after clicking and opening it.	**INT**	- Likes the concept of the game.	**GEN**
		- Player can't tell where to click to start a minigame. Example: having hard time finding where to click to start observation minigame.	**INT**	- Feels that the minigames are too long. Suggests having to finish 1 minigame per area instead of 2.	**GEN**
		- BUG: Mute button in Interview Simulator minigame mutes the main game when returning.	**MGM**	- Enjoyed the puzzle-solving part in the supervisor's office to find the crystals.	**INT**
		- BUG: Background music from some minigames keeps playing when returning to main game.	**MGM**	- Suggests including some tips/help on what to do to progress.	**GEN**
5	**Age Group** 31 - 40 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows of them, but doesn't	- Player attempts to drag items from inventory instead of clicking on them.	**INV**	- Doesn't like having to click a second time after using item from inventory to hide the usage confirmation message.	**DLG**
		- BUG: Toll machine negative feedback on correct use recurring issue.	**INT**	- Thought that Punk NPC was holding a mask in a reference to COVID-19.	**GEN**
		- Player clicks the minigame icons to start minigames.	**INT**	- Found a typo in design-related research minigame story: donat instead of don't.	**DLG**

No.	Subject Information	Observations	Type	Feedback	Type
	actively use them in work nor studies **Time Spent Playing Video Games** Rarely or not at all	- Player clicks on the empty table in front of monster (instead of clicking monster NPC) to give them the game board.	**INT**		
		- BUG: Sound bugs of some minigames' BGM continuing to play in main game.	**MGM**		
6	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows of them, but doesn't actively use them in work nor studies **Time Spent Playing Video Games** Occasionally - A few times a month	- BUG: Toll machine negative feedback on correct use recurring issue.	**INT**	- Doesn't like having to click a second time after using item from inventory to hide the usage confirmation message. Suggests maybe a pop-up instead.	**DLG**
				- Would like to be able to switch the left mouse button and right mouse button controls.	**GEN**
				- Cursor images pointing right and left on screen edges for turning could be bigger.	**GEN**
				- Highlighting interactable objects would be good.	**INT**
				- Including some form of tutorial in the beginning.	**GEN**
				- Suggests using numbers on keyboard to quickly choose dialogue options.	**DLG**

No.	Subject Information	Observations	Type	Feedback	Type
				- Suggest adding sound feedback along with the visual cue when hovering over interactable objects to help near-sighted players.	**INT**
				- Suggests adding more items to be picked up that not necessarily have to be used, but are there to throw-off the player and make them think which item should be used at any given situation.	**GEN**
				- Suggests adding some easter eggs, random facts, and jokes.	**GEN**
7	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows of them, but doesn't actively use them in work nor studies **Time Spent Playing Video Games** Rarely or not at all	- Player tried to drag items from inventory to use them.	**INV**	- Found some typos in research methodology dialogue minigames.	**DLG**
		- Player took some time to notice the 'Return' button on the lock box interface. Instead clicks outside the graphic to exit or clicks the lock box handle after unlocking it.	**INT**	- When making a dialogue choice (research methods stories), would like to be able to return to the options selection instead of having to stick with the selected option till the end.	**DLG**
		- BUG: Cabinet in supervisor's office error message on correct usage (recurring issue).	**INT**	- Participatory Loop minigame has no menu/exit button after starting the game.	**MGM**
		- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Participatory Loop minigame: suggests a smaller number of loops before client character joins meeting.	**MGM**

No.	Subject Information	Observations	Type	Feedback	Type
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- Felt that some interactions have hidden/unexpected consequences. For example, clicking doors starts minigames right away and clicking dialogue space in Persona minigame advances to next screen. Would like a confirmation message before that.	**INT**
		- BUG: The same research methodology minigame can be played in both dimension after one is already won. Noted with design-related research story.	**MGM**	- Felt that the game has too much text.	**GEN**
		- Participatory Loop minigame: Player thinks they can only invite 1 person to the meeting.	**MGM**	- Didn't know that the game has saving function.	**SAV**
				- Would like more feedback about game progression and what to do.	**GEN**
8	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Intermediate **Familiarity with Research Methods** None **Time Spent Playing Video Games** Regularly - Several times a	- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Would like a tip about location of inventory bar.	**INV**
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- White dialogue text on white (quantitative world) background is too hard to read.	**DLG**
				- Some typos in dialogues.	**DLG**
				- Felt that the top menu (inventory) is not used much.	**GEN**

No.	Subject Information	Observations	Type	Feedback	Type
	week			- BUG: The map next to Map Maker NPC shows an interaction, but nothing happens when clicking it.	**INT**
				- Observation minigame: the people are too small to know what activity they're doing.	**MGM**
9	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** None **Time Spent Playing Video Games** Regularly - Several times a week	- Was able to find the inventory right away. When asked, said it's from previous experience with games of the genre.	**INV**	- Would like some feedback on what is clickable/interactable in the main game. When asked about circle on cursor, replied that they didn't notice it.	**INT**
		- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Some minigames are too easy to finish, like the observation minigame.	**GEN**
		- Skipped playing Questioniser minigame because the 'How to Play' had too much info and made them think the game is too complex.	**MGM**	- Does not like having to click twice after using item to hide the confirmation message. Would prefer if it disappeared automatically and didn't block interaction.	**DLG**
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**		
		- BUG: Survey research method minigame/story: Player selected a dialogue option and had no text response from NPC (empty screen), only 2 choices buttons displayed.	**DLG**		
		- BUG: The same research methodology minigame can be played in both dimension after one is already won. Noted with design-related research story.	**MGM**		

No.	Subject Information	Observations	Type	Feedback	Type
10	**Age Group** 31 - 40 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows and actively uses them in work and/or studies **Time Spent Playing Video Games** Regularly - Several times a week	- Player attempted to drag items from inventory bar instead of clicking them.	**INV**	- Doesn't like to read too much and would prefer voice-over with dialogue.	**DLG**
		- Player didn't notice the 'Return' button in the lock box interface. Clicked outside graphic and lock box handle bar when finished. Needed to be told to click the 'Return' button.	**INT**	- Would like some clues or hints about inventory and gameplay.	**GEN**
		- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Liked the circle on the cursor to indicate interactable objects.	**INT**
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- Had no clue that they can turn around in 2nd level (Act 2).	**INT**
		- BUG: The same research methodology minigame can be played in both dimension after one is already won. Noted with case study research story.	**MGM**		
11	**Age Group** Under 20 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent **Familiarity with Research Methods** Knows and actively uses them	- BUG: Dialogue breaks first time speaking with Map Maker (recurring issue).	**DLG**	- Would like hints about what to do in the game to progress.	**GEN**
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- Felt that the game is quite big	**GEN**

No.	Subject Information	Observations	Type	Feedback	Type
	in work and/or studies **Time Spent Playing Video Games** Rarely or not at all				
12	**Age Group** 20 - 30 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Intermediate **Familiarity with Research Methods** Knows and actively uses them in work and/or studies **Time Spent Playing Video Games** Rarely or not at all	- Player attempted to drag items from inventory bar instead of clicking them.	**INV**	- Would like more hints about the overall game story. Felt that the story wasn't clear enough. When asked, didn't read much of the dialogues.	**GEN**
		- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- Would like a hint about ability to turn around in second level.	**INT**
				- Felt that the minigames are good and able to teach something about research methods.	**GEN**
				- Suggests making the game requirements as clear as possible for everybody. As some researcher scholars playing the game may have no experience with games.	**GEN**
13	**Age Group** 31 - 40 **Education Level** Higher Education - University student or graduate **English Lang. Proficiency** Fluent	- BUG: Cabinet in supervisor's office error message on correct usage (recurring issue).	**INT**	- Didn't like that they couldn't interact with objects when the confirmation dialogue message comes up (after using an item).	**DLG**
		- Attempts to click on the gate after using the toll machine to open it, instead of clicking the toll machine.	**INT**	- Felt like there's a lot of reading.	**GEN**

No.	Subject Information	Observations	Type	Feedback	Type
	Familiarity with Research Methods None **Time Spent Playing Video Games** Rarely or not at all	- BUG: Toll machine negative feedback on correct use (recurring issue).	**INT**	- The dialogue text on an image background is hard to read. Suggests putting it in solid background box.	**DLG**
		- BUG: ProtoEscape minigame: During alien review/feedback, and after clicking 'Next' button to the end of the feedback, the 'Skip/Complete' doesn't work and the game needed to be restarted.	**MGM**	- ProtoEscape minigame: > Felt that it is difficult to understand what to do. The alien puts their hand on a random thing every time they give feedback. > Player assumed all the journal entries were only for the first screen/box review. > Would like a tutorial on how to play the game. > Thinks there's too much text in the journal, and would prefer more graphics.	**MGM**
		- BUG: Research Papers minigame: A paper review comes up from previous level after it was ended and before starting the next level.	**MGM**	- Interview Simulator minigame: The pros/cons at the end did not relate to the choices they made during the game.	**MGM**
				- Research Papers minigame: > Instruction book lists rules by number (1, 2, etc.), while the screen and buttons are numbered by letters (a, b, etc.). Prefer all to have the same listing style. > The auditor sometimes gives feedback even when the player made the correct choice or rejecting a paper. Made them think they did a mistake.	**MGM**
				- Felt that all minigames were confusing with their goals. However, mentioned that it could be because their lack of knowledge of research methods.	**MGM**

No.	Subject Information	Observations	Type	Feedback	Type
				- Felt that main game dialogues were too long. Prefers short and simple ones like in ProtoEscape minigame.	**DLG**
				- Felt that the dialogue options (main game) were too many, and that it was hard to keep track of which options were selected before.	**DLG**