

# **Computer Science Senior Design Final Report: Team 116**

Sidney Zweifel (Team leader)

Gretta Foxx

Yaxin Gu

Sarah Ogorzaly

Team 116

Department Computer Science, Cleveland State University

Submitted to—

Professor Robert Fiske, Faculty Advisor

Department of Computer Science, Cleveland State University

Dr. Sanchita Mal-Sarkar, EEC 493 Professor

Department of Computer Science, Cleveland State University

## Contents

Executive Summary	4
1. Statement of Problem and Background	7
2. Design Objectives	8
3. Technical Approach	8
3.1. Identifying the Unmet Needs	8
3.2. Determining the Design Constraints	9
3.3. Defining the Technical Specifications	9
3.4. Enumerating the Design Concepts	10
3.5. Selecting Design Concepts	10
3.6. Standards Compliance	11
4. Detailed Design	12
4.1. Scene Manager	12
4.2. Global Randomization	14
4.3. Level One	15
4.4. Level Two	17
4.5. Level Three	18
4.6. Level Four	21
5. Verification	28
5.1. Testing Level One	28
5.2. Testing Level Two	28
5.3. Testing Level Three	29
5.4. Testing Level Four	29

6. Project Management	31
6.1. Team Qualifications	31
6.2. Task Assignment	33
6.3. Timeline	36
6.4. Deliverables	37
6.5. Budget	37
6.6. Communication and Coordination with Sponsor	38
7. Professional Awareness	39
8. Conclusion	40
9. References	42
Appendix A: Résumés of Team Members	43
Gretta Foxx	43
Yaxin Gu	44
Sarah Ogorzaly	45
Sidney Zweifel	46

## Executive Summary

In education, computer science is a rigorous subject with many complexities and copious amounts of material to learn. It is not uncommon for Cleveland State University students to struggle when initially learning new topics and programming concepts. That struggle can feel even more difficult when these new concepts are not being taught in an engaging and intuitive way. That is why our design group proposes an educational computer science game in which a player is put into an *escape room* where they are tasked with solving computer science-themed puzzles to escape and beat the game. Ultimately, we want our game to be a resource that students can turn to learn and reinforce important aspects of computer science. The content that will be covered in the game is reflective of the core concepts taught in Computer Information Science (CIS) courses at CSU. Some of these concepts include but are not limited to assigning variables, if-else statements, bitwise operation, binary conversion, loops, arrays, objects, inheritance, data structures, and algorithms. We developed our game with Godot, an open-source game development software. We chose this engine so that future developers, students, or faculty can easily alter or add to the game based on their specific needs. On top of making the game design adaptable, we developed the game with thoughts of making gameplay intuitive to gamers and programmers at all levels. To do this, we conducted research on how to make gameplay fluid and consistent in player interaction while also ensuring that the game is efficient and replayable. This replay value aspect adds a technical edge to our project as it required us to build each of the puzzles in a way that ensures comprehension rather than memorization. One of the primary ways this was accomplished was with randomization. Additionally, we aimed to have a strong creative and visually appealing aspect to our project since we are a team inspired by bridging art and STEM. We collaborated on game design, implementation, and testing to ensure that our project meets the objectives that we set in place. In pursuit of project objectives, we planned out an organized technical approach plan that aligns with standard compliances. Our final deliverables include an educational game that will help students improve their programming skills in a fun and engaging manner as well as organized and coherent game documentation.

### 1. Statement of Problem and Background

Video games are often only thought of as activities of leisure. A hobby that allows people to relax or escape to a different world for some time. An overlooked subset under the gaming umbrella is educational games. Educational games provide an experience curated toward teaching the player a skill that can be fully transferable into the real world. These video games can be very special when they are both engaging and informative as they will encourage the player to want to learn in a self-directed way [1], which gives learners confidence when working with unfamiliar concepts since it allows them to practice in a relaxed learning environment where they can make progress at their own pace. Educational games are a valuable resource that should be unitized by universities because they enhance the learning experience by simulating

traditional ways of learning, are experiences specially built for developing problem solving skills, showcase the real-world experience of working on multi-disciplinary projects, and inspire future students to build their own projects.

To prove the assertion that a video game is a perfect fit for learning, consider that students and players both use the same problem-solving techniques. As a first step, they both start with obstacles they must overcome to achieve a goal [2]. For students, this goal might be mastering polymorphism. For a player, the goal might be searching for the code to unlock a safe. Either way, each party has a set intention that they will attempt to achieve.

After a goal has been established, each party enters a stage where they are interpreting their previous experiences and thinking about what might be useful from what they already know [2]. A common study practice for the student would be to look at examples of polymorphism. Notice, however, that it would be difficult for the student to fully comprehend polymorphism without a solid grasp of objects. This is why the student must rely on their previous knowledge as a basis to learn more about a similar subject. A player, on the other hand, would perform a thorough search of their environment by searching for clues through drawers and interacting with playable items. The player's search is guided by their assumptions of where codes are usually put in similar games and their real-world knowledge of how humans hide things. Despite the actions of the student and the player seeming unrelated, they are both recalling what previous actions have shown them to provide a suitable foundation for their discoveries [1]-[3].

The next step is perhaps the most important, based on their actions, they get immediate feedback that determines if the goal was achieved. [2]. The students get feedback when they attempt to fully explain a subject and see the gaps in their knowledge. This allows them to continuously recall their knowledge in different ways and pinpoint what concepts they should review more. The learning process of the player is a much more streamlined process. Once they do not find what they are looking for in one area, they quickly move on to the next. However, because the player is exploring their environment, they are gaining knowledge that will prove useful to them later such as the location of a suspicious-looking book or even how to use certain game mechanics to their advantage, such as a certain sound means that there is a hint nearby.

This loop of learning something new, attempting to apply that knowledge, and getting feedback, is added to the student's and player's memory bank of experiences, allowing them to interpret the best combinations of actions to take in the future [2]. Once the parties continuously reach their goals, they have successfully learned a new skill [3]. These newly learned skills will then give them a basis to extend their knowledge to greater heights.

Where gaming rises above a student's learning traditionally is in its medium. Educational games are curated toward teaching the player skills. The student does not need to sift through a

heap of unhelpful information to find what they are looking for, because the game is created in a way where they develop the necessary skills, they need to solve the obstacles and reach their goals. Furthermore, a video game is a multi-sensory experience. This gives players a more engaging experience because they are interacting with a simulated environment with its own objects, colors, sounds, and music. A puzzle game is especially helpful for computer science students because being an excellent problem solver is essential. Students can only *level up* their problem-solving skills if they are regularly solving problems which require them to understand the patterns of problem-solving and how they synthesize what they already know to reach new conclusions.

Another vital reason that this project will be worthwhile is that it fills the need for an educational game made at CSU. As CSU moves toward integrated STEM degrees, it is valuable to have a project to showcase that conveys the importance of people of different academic backgrounds working together to create something that showcases the combined talents of others from different fields. Our project is only the beginning of what could be something that many CSU students get a chance to work on. Other students can test and learn from their creations. It could be a cycle of people learning from created games, getting inspired to build their own projects, and having others learn and be inspired by their projects.

To summarize, we first asserted that video games are closely related to the learning process and provide a specifically curated experience where all elements of the game serve to keep the player engaged. We then explained that our project not only meets the need of beginner computer science students to expand their knowledge and advance their problem-solving skills in a fun way, but also aligns with many of the goals of CSU. In the university's effort to prepare students for the real-world experience, our project portrays how a team can combine their technical knowledge with their soft skills, artistry, and creativity. Beyond these points, because the primary goal of this project is to provide easy to comprehend documentation; this project is meant to be the beginning of something that any CSU students have a chance to be involved in if they are so inspired. Many students are interested in pursuing game design and what better way for them to get experience with game design and advance their degree than developing more levels for a game that explores the pivotal elements of what they are studying?

## 2. Design Objectives

This document discusses the development of an educational computer science game inspired by escape rooms. An escape room [4] is a hands-on adventure game in which groups work together to “escape” from a room by solving puzzles before time runs out. Our design objectives are noted as follows:

1. Successfully make a PC game in Godot that uses multiple “escape room”-esque levels to teach people important computer science concepts in an engaging way
2. Create a scene manager for the game that assists in providing players a replayable experience with randomization while also ensuring smooth transition between levels to ensure they understand concepts rather than memorizing answers
3. Develop a game with clear documentation and instructions so that it may be easily built upon by creating new levels or modifying existing ones to meet student and faculty needs
4. Successfully work in a team of individuals inspired by blending the boundaries of STEAM (Science, Technology, Engineering, Arts, Mathematics)

The game we developed includes multiple escape rooms to represent game levels. The difficulty of each room/level will be based on different concepts taught in [5] CIS courses at CSU. For example, level one and two will focus on basic programming ideas taught in CIS 151. Therefore, the puzzles include concepts such as assigning variables and if-else statements. The same applies to level three and four as the puzzles focus more on content covered in CIS 260, such as bitwise operation, binary-to-decimal conversion, and loops.

An essential part of these game levels will be the randomization aspect. We made all our puzzles random with each new play. This means that puzzles will not always have the same answer despite the puzzle design remaining the same. For example, if we had an activity that involved solving logic problems to get a number code— the numbers of that code would be different each time. This reinforces learning computer science concepts since the player cannot simply rely on memorizing answers to beat puzzles. Players will need to understand the underlying logic of the puzzle to advance to the next level. Additionally, this will give our game better replay value [6] because it will keep the game more interesting when players encounter new problems on each replay.

While we do have many ideas for different levels of our game, it is likely that we will not have enough time to complete everything we have planned. This is why we created very clear and thorough documentation of how the game works from a developer’s perspective. The same applies to instructions on how to play the game from a player’s perspective. This easy-to-read documentation serves as an entry point for future developers, faculty, and students to modify or add to our game. Overall, we wanted to make it intuitive for anyone to modify our existing levels

or create new levels for their specific needs. We note that this will also require us to make our code as coherent as possible with sufficient commenting.

All that we have accomplished has been dependent upon being strong in team collaboration. Our group members all have similar interests in the fields of game development, software engineering, and the arts. With our combined professional interests and computer science experience, we completed a project that is both meaningful and reflective of our abilities.

### **3. Technical Approach**

#### **3.1. Identifying the Unmet Needs**

We identified the unmet needs of the customers by examining the learning experiences of computer science students. There are many people learning computer science, including many students at CSU. However, the concepts can be complex and hard to understand, causing students to struggle during the learning process. This gives rise to the need for a simpler and more engaging method to learn the concepts. Our educational video game strives to meet this need. Our game is based on solving puzzles in escape rooms. Applying computer science concepts to the puzzles makes learning an interesting activity. Our game is based on computer science courses offered at CSU, making it more relatable to CSU students. We focused on breaking down difficult concepts into simple puzzles that students with different levels of experience can easily understand.

There is also a need for the game to have strong replay value. The game aims to be more entertaining and helpful to play if the game has variables that change during each playthrough. Our second objective was to create an engine for the game to add randomization to the game variables so no puzzle will be the same. The puzzles can be solved with the same method, but because we used random variables, the player will have to solve the puzzle each time instead of memorizing the answer. Students will be able to learn the concepts better by practicing with different variations of the puzzle. Randomization will also be more fun and challenging than just repeating the same puzzle.

The game will be better suited to students' needs if students implement their own customizations and build upon the game. Each student's learning progress will differ so the game would be most useful if students can modify the game to their needs. Since we are finished with development of the game, it will be available for modification by other users through our GitHub repository. We provided clear documentation and code commenting of the game so others can easily pick up from where we left off. Students and faculty can modify the existing levels or create new ones. This way the game can continue to be developed and more people will be able to utilize it as a learning tool.



### **3.2. Determining the Design Constraints**

The design constraints imposed on the project include a lack of experience working with the software. We used Godot to create our game, which is a software we only recently started learning during the beginning of the Senior Design course. Despite using it to create a few simple games, there were still new aspects of Godot to be learned throughout the development of our game. This did limit our ability to implement advanced game mechanics in some sense as we spent a substantial amount of time figuring out the best approach to handle any issues we encountered. It was also a challenge to develop levels for complex concepts. We did not make as much progress as we hoped for on puzzles dealing with more complex topics due to time constraints and needing more time to learn how to deal with more advanced game mechanics.

Another constraint was the lack of version control software. For file sharing and collaboration, we used a GitHub repository. We have experience using it, but it is more prone to making mistakes and management can be difficult. We are currently using GitHub branches, with each team member creating their own branch for editing the project. The individual branches were tested before changes were committed to the main branch. This was a tedious process in which we often ran into merge conflicts. We had to pay close attention when making changes to or merging a branch.

Finally, tackling the game scene manager's need for randomized functionality and seamless transitions proved to be the biggest challenge. This required the creation of a dedicated game scene and script tailored precisely for dynamically altering puzzle elements. Particularly challenging were intricate concepts like algorithms, which necessitated a more intricate approach. Initially, we planned to utilize XML files for configuration, but through experimentation, we discovered that Godot's built-in randomization tools paired with JSON files better aligned with our requirements. As randomization stands as a cornerstone of our objectives, mastering its implementation was essential.

### **3.3. Defining Technical Specifications**

We determined our specifications by constructing a list of essential elements that we believe allow our game to be an educational and engaging experience. First, we have successfully created our PC game using the Godot game engine in which our game is able to run and operate with no crashes or game-breaking bugs. We wanted to focus on this because we believe that if our players are struggling to play the game, it will be almost impossible for them to learn anything meaningful. For the gameplay, there are 4 "escape-room"-esque levels. We unfortunately did not have the time to implement more levels although that could be a way we could expand upon our game in the future. Each level provides a learning experience taught through a unique puzzle, with each level focusing on a different computer science topic. The topics explored in each level increase in complexity as a player progresses through each of the levels. This gradual progression allows the players to be eased into complex topics, mitigating the risk that they get confused or overwhelmed.

Our four levels also have some aspects of replayability that allow for multiple playthroughs. We wanted to focus on this especially to ensure that our players can truly understand concepts rather than just memorizing the answers. In addition, we have included documentation that makes it simple to create new levels for our game. This documentation can be used by CSU faculty members to teach their students using our game, or by students as a jumping off point for beginning their own game development endeavors. Despite some differences in difficulty between levels, we believe that our completed game is a robust, replayable, and educational gaming experience that is successfully able to teach students our selected concepts of computer science.

### **3.4. Enumerating Design Concepts**

Our first design concept was to have one group member tackle one computer science concept per level, totaling four topics total covered in our game with each group member implementing their own level and concept. While we all have worked together on the broad aspects of our game, each group member has developed their own puzzles for a concept of their choice.

Our second design concept is the replayability of the puzzles in our game. We think that rather than only memorizing the answers to a single question, players will learn more about a certain idea if they may revisit our levels with new numbers or variables. If a player is not confident about a given topic, they can replay a level as many times as it takes to understand it. Each time they restart the level, some aspects of our puzzles will be different. For example, in the puzzle introducing the concept of a for-loop, the range changes or the numbers in the problem are different, so on the first playthrough the starting index could be  $i = 5$ , and the second time  $i = 33$ . Overall, we want players to understand the underlying logic of a puzzle rather than the solution to the puzzle.

Another design concept is the addition of hints in each puzzle. These hints serve to give the players clues on what their next step should be and, in some cases, show them how a computer science topic works, such as conversion of binary numbers to integers. These hints are used to guide the player and keep them engaged in the gameplay. The goal is to encourage players to keep trying, despite the difficulty they may face.

Our final design concept is to gradually implement more advanced concepts as the levels progress. The first level starts with the very basic concept of variables and print statements. The second level introduces the more complex concepts of if/else statements. The third level covers bitwise operations using binary numbers and the last level covers the concept of for-loops.

### **3.5. Selecting Design Concepts**

The design concepts were developed based on our own personal experiences in our computer science courses at CSU and our own experiences with creating simple Godot games.

The team started off by constructing a list of the computer science concepts that were the hardest for us to grasp when first learning them in the courses we have taken at CSU. We used that to create the list of topics we wanted to cover. We decided that we would want to give our players an experience similar to learning where it starts at the fundamentals and there is a gradual progression into complex topics.

Our initial design aimed to craft an escape room game featuring three distinct levels, each comprising four sections centered around diverse computer science concepts. We opted for this structure with the intention of guiding students through progressively challenging material. Although time constraints prevented us from completing our original vision of three expansive levels with four sections each, we remained committed to the core idea of escalating puzzle difficulty as the game progressed.

By breaking the final game into a total of four distinct sections, or levels, we not only evenly distributed the workload among group members but also afforded creative freedom to each team member. This division ensured that each member could contribute their puzzle design without hindrance, fostering an environment where progress was not contingent on others. Thus, our setup allows all team members to advance at their own pace.

Additionally, the randomization feature was employed to enhance replayability by generating random numbers upon a new game session being started. This ensures that each playthrough offers a unique experience. We utilized randomization to generate values such as bitwise operation problems in level three and to establish varying loop boundaries in level four. While time constraints limited the development of more intricate puzzles, we envisioned future levels delving into advanced concepts like complex algorithms, multi-threading, and machine learning. Implementing these concepts is anticipated to pose greater challenges, amplifying the need for a more sophisticated randomization tool.

We considered an alternative design approach aimed at enhancing the game's versatility by enabling user modification. This concept would empower users to tailor various aspects of the game to suit their preferences and needs. From adjusting generated numbers to modifying puzzle logic or even adding entirely new levels, users are afforded significant flexibility. To support this, we meticulously documented the game, ensuring that other users can easily comprehend and modify its components according to their desires.

### **3.6. Standards Compliance**

The development of educational video games involves compliance with industrial standards and government regulations. One applicable standard is ISO/IEC 25010. It is a quality model that determines the quality aspects of a software product. The qualities include functionality, performance, compatibility, usability, reliability, security, maintainability, and portability [7]. Some qualities that are applicable to our project are functionality and

maintainability. Functionality quality ensures that the game functions effectively contribute to the learning objectives. Our project design of puzzles and level-based systems should effectively teach players the concepts in an engaging way. Reliability quality assesses the stability and consistency of the game. We complied with the standard by making sure the game operates smoothly and without crashes. Maintainability quality considers the modifiability and reusability of a software. A key part of our project objectives was to provide clear documentation for easy modification and to provide randomization for replayability.

A government regulation that is applicable to our project is copyright and intellectual property law. Some assets that will be used in our game, such as background music, will come from online sources. We will make sure to give credit to the creator when using any assets that are not royalty-free.

## **4. Detailed Design**

### **4.1. Scene Manager**

The scene manager was our first major goal during development. It is a global script that determines when and whether the player can advance to the next puzzle or not. It uses a passcode-based system. If the player enters the correct passcode, they may advance to the next level. If the player enters the passcode incorrectly or simply closes out of the window to enter the passcode, they are forced to restart the level or at the very least have their position reset to the start of the puzzle. This design of restarting the level due to incorrect answers was to increase the challenge of the game by rewarding players that took the time to learn the concepts and encouraging struggling players to try again. Although, most of us opted for some mercy during level design and added checks in so that the player would not have to restart the entire level.

Before getting any further into development, it is significant to note that during the development of the scene manager two main videos were taken reference of [11] and [12]. Their code and ideas guided us in developing our scene manager but were not directly copied. Also note that a complete list of references is provided in the documentation. The most essential aspect taught during both videos was how the export keyword worked in Godot —as a way to easily enter values in the inspector interface on the right-hand side of the screen.

The first major challenge to overcome regarding the scene manager was thinking about how the game would know which scene to load next in a more intuitive way than directly programming the changing of scenes. To make the scene manager as general as possible, we opted for creating a door asset to control the changing of scenes.

The door asset is made up of an image of a door and has a button node on top of it so that it may be clicked and open the passcode scene. To get implement the logic of opening the passcode outlined above, we require that each scene that the door is instantiated in require the following three exported variables: the path to the new scene, the path to the passcode scene, and the path to the current puzzle. This ensures that when the door is clicked the game knows where

to put the player. Note the even though the path to the passcode scene is essentially the same for each level, the door requires the scene to be reimplemented for each scene that the player uses the door. This could be improved upon quite easily by directly putting the path to the passcode scene in the scene manger autoload but was also an intentional choice because it allows the team to put anything scene want in the passcode scene and still be able to use the scene manager as intended. In fact, this feature is an essential part of the Level Four design where the passcode scene is replaced be a different graphical interface.

As mentioned before, exporting these variables is helpful as Godot provides a place for the developer to input the required information right in the user interface of the editor. It makes changing the scenes a simple process. All in all, the door provides a way to cleverly use the scene manager because it requires that the programmer enter the required information that the scene manager needs with an asset that all the levels need to advance onward.

The scene manager itself is in the *global.gd* script. For our scene manager to work it must be auto loaded in Godot, which means it must be always loaded when the game is running. Other nodes such as the door are loaded when the scene that they are a child of are loaded. In our global script, several variables are defined for global use in the script. These include the path to the next puzzle, the path of the current puzzle, and the current passcode.

The first function in the scene manager is the *handle\_door()* function. This will determine if the player advances to the next puzzle or be reloaded in the current puzzle. In Godot, everything is works of a tree structure, so if a parent scene is freed then all the children of the parent are also freed. Furthermore, because it is a tree structure it would be an erroneous to have the current scene be a child of itself. In other words, it would be an issue if the scene was changed before the current scene would be freed especially if the new scene was the same as the old scene. This is why the current scene is saved and freed from the tree before the scenes are changed.

The *go\_to\_passcode()* function is a simple function that takes in the information from exported information from the door (i.e. the paths to the scenes of the current puzzle, the next puzzle, and the passcode puzzle) and sets them accordingly to the global variables. Moreover, it also changes the scene to the passcode scene hence its name. The next function, called *return\_to\_puzzle()* frees the current scene from the scene and changes the scene to whatever path was added as the current puzzle. As is about to be mentioned in the paragraph below, this function could benefit from establishing a system to keep desired player and level information so that the level is not reset each time and each of us would have saved time programming ways to save the player progress.

There are some limitations with the scene manager. One of the greatest is that the player data is not saved when transitioning between scenes. This means that the when the player enters a passcode scene and does not advance to the next scene (from either closing out of the passcode scene or entering an incorrect passcode) that the puzzle reloads again. The most noticeable part about this drawback is that the player's position is reset to where they were at the start of the

level, which makes for a clumsy game design. Although, this bug can be used as feature, such as in Level Four where these resets add to the theme of the puzzle's theme of looping.

Moreover, the decision to make the door and passcode so intertwined proved somewhat problematic during the development of the puzzles. When the puzzle resets it also resets the password each time the player got the passcode incorrect making the game difficult to test and play because too many elements were changing at once. As such changes were made to the scene manager to accommodate the logic of our specific puzzles, but these changes were solely additions to the scene manager and the underlying logic is still there. If we were to develop the scene manager again, more priority would go to developing a method how to save player data and generating the passcode to leave only when it is necessary to the specific puzzle.

## 4.2. Global Randomization

As has already stated a multitude of times, randomization is one of the core features of our game. The specific of the randomization of each of our levels is noted in the specific Level section. This section explores the randomization of the passcode generation.

As explained above, our Scene Manager works with a global script, which in Godot is called an Autoload. This script is always loaded when the game is running and can therefore be called upon in any part of the game. In our global script entitled `global.gd`, two functions deal with the passcode of the door for each level: `set_passcode()` and `get_passcode()`.

In `set_passcode()`, a random five-digit code is generated by using Godot's built-in random number generator for producing random integers between a range. The range given to the function is 10000 to 99999, note that in Godot this range is inclusive for both the lower bound and higher bound. This range is built to always provide the player with a five-digit code. The decision for the passcode being five-digits is arbitrary. It was simply a number to set the number of digits that the passcode interface could take before it determined whether the entered passcode was correct or not. Although, it should be noted that the passcode can deal with less digits if the user hits enter before putting in five digits.

Moving on to the implementation, the `set_passcode()` function is called in the `passcode.gd` file which lives in the `Popup/Passcode` folder in our project's file hierarchy. This script is for the behavior of the `Pop_Up_Test` scene. The purpose of this scene was for testing and debugging the passcode randomization behavior. It provides a button to the player that when clicked will reveal a popup that tells the player the current passcode. The reason why it sets the passcode is because the current passcode needs to be set before the scene that checks if what the user entered was correct. In theory, `set_passcode()` could be called in the same scene as `get_passcode()` is but this was implemented very early in project development and fixing other features took priority. In theory making this quick change should not cause any issues, but in this late in development we opt to not change features that are not necessarily broken.

In the *get\_passcode()* function the current set passcode is simply returned to whatever called the function. In the current implementation this function is called in the *Passcode\_Enter* scene, this scene lives in the *Popup/Passcode* folder as well. This scene's job is to provide the player with a visual interface to enter the passcode. When the player starts typing a potential passcode a signal is emitted that takes in the user input. It is when this signal is emitted that the *get\_passcode()* function is called and the user-entered value is checked against the current randomized passcode. Also provided in this script are other calls to global that handle whether the player may advance to the next level or not.

Again, this part of the code is admittedly not the best way to implement everything. It was our plan to go back and redefine this code and scene manager after we had implemented our puzzles, but we ran short of time. Now, looking back at the project development, and how each of us had to develop our own methods to change the behavior of the passcode to what we needed in our puzzles; it would have been advantageous to work together on a solution that worked for all of us. Perhaps the passcode generation could have been individually implemented in each puzzle or maybe puzzle design should have been concrete before determining how the passcode generation would work. Nevertheless, this part of the project serves as a reminder of growth in learning how game development works in Godot and how to manage a team more effectively.

### 4.3. Level One

The first level takes place when our player character wakes up in their cell and covers the basic topic of how to print variables. The goal of the puzzle is to obtain the passcode for the door which can be found by digging through the files that are available on the terminal to the left of the room. The player will first wake up and can search around their room. There will be a hint that the player can read that will point them to look through the terminal as well as draw their attention to the color of the light by the door. Each time the game runs, the color of the door is randomized to be one of 5 different colors. Each door color will then correspond to a specific door number that will also be randomized to be one of 5 different door numbers.

The computer science topic this first level is trying to teach is how to print out variables and traverse a mock file system. The variables that the player can print out that will be relevant to the puzzle will be door 1 color, door 2 color, door 3 color, door 4 color, door 5 color, door 1 code, door 2 code, door 3 code, door 4 code, and door 5 code. There will also be other variables that the player can print out that will contain story details for our game, but they will not affect how the puzzle functions or how it will be solved.

The door code is solved by first having the player note the color of the door light. They will then look through the terminal and see that specific door numbers go with each door light. Once they figure out what door number corresponds to the color they have by their door, they can print out the door code that opens the door. For example, if door 3 has a yellow light, then they will have to print out door 3's passcode to open the door. The typical gameplay loop can be seen in the images below.

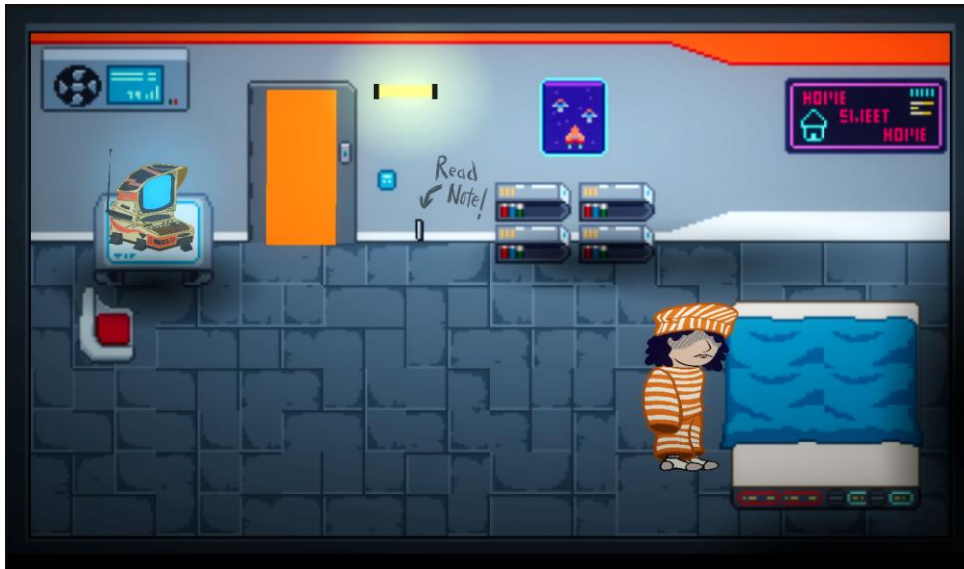


Figure 1. Shows door color is yellow in level one

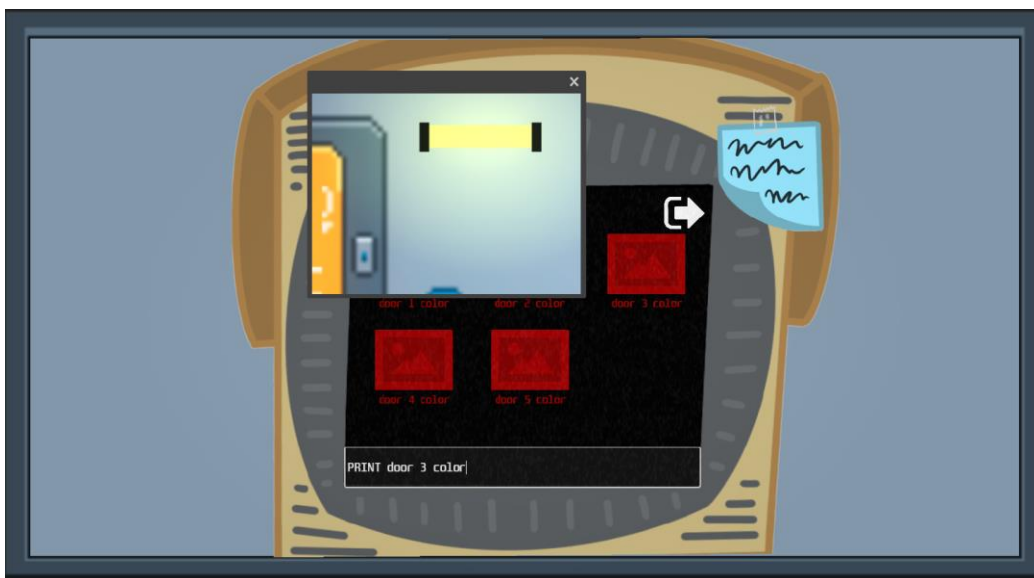
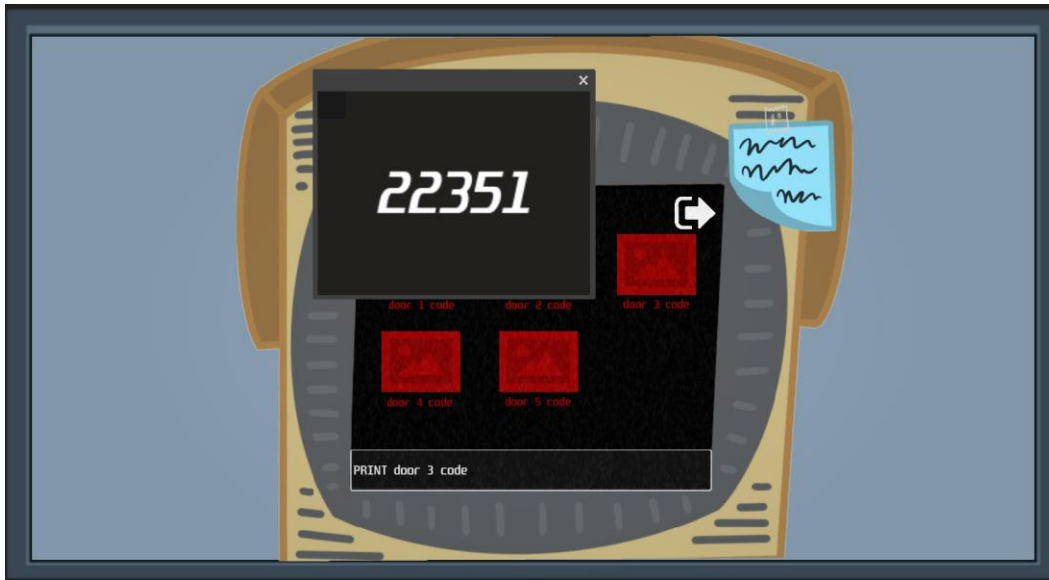


Figure 2. Printing out door color in level one, yellow is door 3





*Figure 3. Printing out the door code that allows the player to progress to level two*

The “terminal” works by having each page of the terminal be a different scene. So, when the player enters a folder, the screen will change to a different scene, that is the representation of inside that folder. This gives the player the illusion they are moving through a file system. When the player prints a variable, a pop up appears with contents of that variable that gives the illusion that the player is printing it. The door number/light pairs are generated when the game starts using the autoloading global script. The *Start.gd* script that is associated to the *Menu* scene calls these functions. First, it calls *Global.set\_door\_passcodes()* to set the passcodes for each door code variable that can be printed in the terminal (door code 1, door code 2, etc.). This generates a random 5-digit passcode for each door 1–5. These passcodes don’t do anything, but they give the illusion that each door has its own unique passcode. It’s important to note that the passcode that allows the player to progress to level two will be reset later, but right now the passcode it is set to doesn’t do anything. Then, the door light is randomly picked from 5 different colors (blue, red, yellow, green, or purple) using *Global.set\_door\_color()*. This color is then paired with a random number 1-5 using *Global.set\_door\_number()*. Lastly, the other door pairs are generated from the remaining colors and numbers using *Global.set\_door\_pairs()*.

#### **4.4. Level Two**

The second level focuses on the concept of if-else statements. The level starts with hints and simple examples to teach players how the statement works. Then the player will have to solve if-else puzzles to make advancements in the game. The puzzles will have statements that need to be changed, such as “if (x = “ ”)”, or statements that need to be changed to true or false. The puzzles will first explain the concept in pseudo codes. Then go over the syntax of if-else statements, nested if-else statements, and else-if statements. The puzzles will have randomization

in variable values used in statements, depending on the condition different statements may be executed.

The level starts with the player stepping outside the cell, granting them the freedom to roam the hallway corridors. Scattered throughout the environment are various objects, like posters, strategically placed to offer hints on solving if-else statements. Meanwhile, robots lurk, poised to capture the player on sight. Being caught results in a reset to the level's start. To progress, players must devise strategies to manipulate the if-else statements governing the robots' behavior, to stop them from chasing the player. Additionally, they will encounter challenges posed by these robots, presenting if-else puzzles that must be solved to acquire the passcode necessary for advancing to the next level.

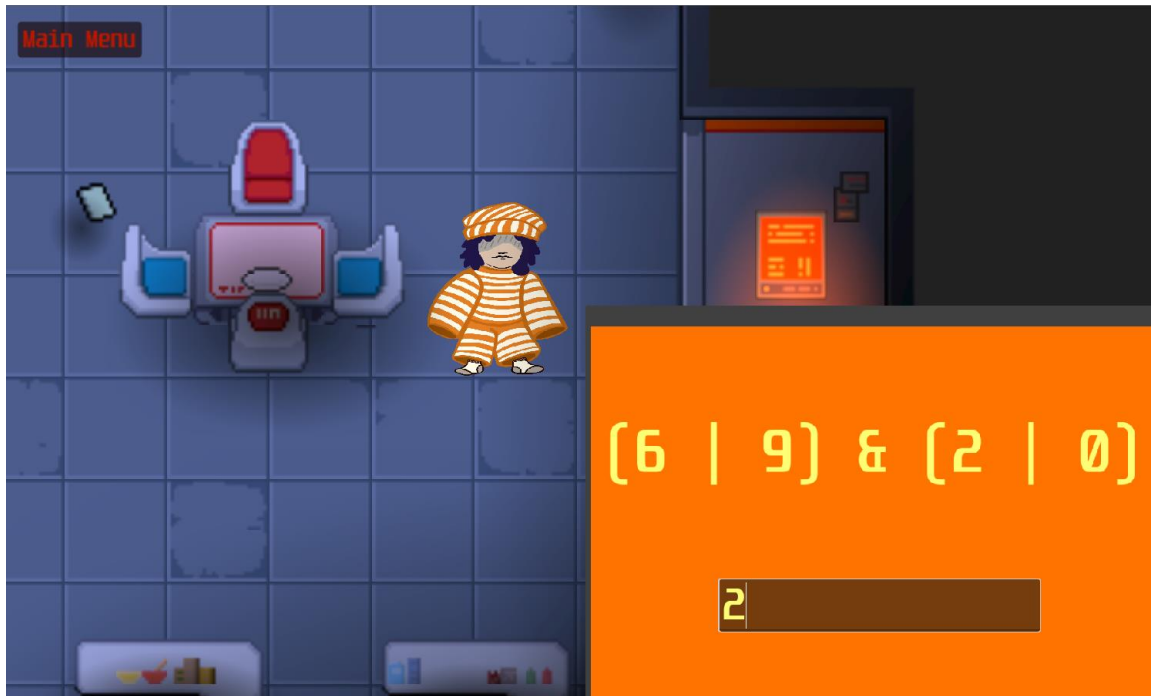
#### 4.5. Level Three

The third level, which takes place in the game's prison cafeteria, focuses on binary and bitwise logic in computer science. The level features three hints, two puzzles, and a passcode activity that allows the player to move to the next room, or level four. For the player to get the correct passcode, they must use the three hints to solve the two puzzles. The three hints, which focus on how to solve bitwise logic operations and binary-to-decimal conversions, are readily available upon unlocking level three. These hints are similar to ambiguous hints and hidden meanings that are often found in escape rooms [4]. The two puzzles must be solved in a specific order as the player will only have access to the binary conversion puzzle after solving the bitwise operation puzzle.

The first puzzle in the game involves the player clicking a button, represented as a computer screen within the game environment. This action leads them to a pop-up window where a random bitwise operation problem is displayed. To solve this puzzle, players must deduce the result of the bitwise operation presented to them and input the answer into a designated line below the operation. This bitwise operation puzzle is managed by both a local and autoloading global script.

The process begins with an autoloading, or global, script using the *set\_logic\_question()* function to generate a random bitwise operation question. The generated question uses AND (&) and OR (|) operators. This function randomly selects integers and logical operators to formulate a question in the format of "(a operator b) operator (c operator d)", storing it for future reference. Upon encountering the puzzle, a local script, attached to the node representing the bitwise operation pop-up, takes charge of user interaction and solution verification. During initialization, this local script retrieves the stored bitwise operation question using the *get\_logic\_question()* function from an autoloading script. As players submit their answers, the local script converts the input into an integer and compares it against the solution obtained from the *solve\_logic\_question()* function in the global script. If the player's answer is incorrect, no action is taken. If the answer is correct, the bitwise operation pop-up window closes, granting players access to the second puzzle.

In figure 5, a test run of level three demonstrates how the bitwise operation puzzle may appear during different game sessions.



*Figure 5. Bitwise operation puzzle in level 3*

The second puzzle involves the player navigating to a different button, represented as another computer screen in the game environment. The puzzle that they initially had no access to is available for viewing. The player is presented with a pop-up window that displays the 5-digit answer to the passcode activity in the form of two different repeated symbols. Through a pop-up hint, the player will need to figure out what the symbols represent. After doing so, the player will find that the symbols represent 1's and 0's and are five different binary numbers that must be converted to decimal. In other words, the player must convert a display of symbols to binary and then convert the binary numbers into decimal to get the passcode that will unlock the next room or the fourth puzzle. This binary conversion puzzle uses both a local and autoloaded script to function.

To initiate the puzzle, the `set_passcode()` function in the autoloaded script generates a random passcode that ensures it falls within a specific range of five-digit numbers. This randomly generated number is the passcode to unlock the next level. Once this passcode is set, a local script attached to the node that displays the symbols to be deciphered is utilized. The local script retrieves the passcode using the `get_passcode()` function from the autoloaded script, subsequently converting the passcode into a binary representation before displaying it symbolically. Each digit of the binary code is represented by either an “eye” symbol for 0's or a “full circle” symbol for 1's. As players interact with the puzzle, they must decipher the binary

passcode to progress. This challenge offers a unique twist, blending visual puzzle-solving with logical deduction.

A test run of level three showcases the varying appearances of the binary conversion puzzle across different gaming sessions, as depicted in Figures 6 and 7, both before and after unlocking access through the bitwise operation puzzle.



*Figure 6. Symbol to binary to decimal conversion puzzle before access in level 3*



*Figure 7. Symbol to binary to decimal conversion puzzle after access in level 3*

If time constraints did not limit us, there are some elements of level three that could be added. One aspect is including XOR (^) operators in the bitwise operation puzzles. Due to Godot confusing the XOR operator with an exponent, a workaround was needed to accurately solve and compare any bitwise operation problems with the XOR operator present. To ensure that the bitwise operation puzzle worked efficiently at every session run, XOR could ultimately not be included. Another aspect that could be added is another puzzle to further complicate level three. This means the player would need to solve three puzzles instead of two to beat level three. Some concepts for this third puzzle include decimal to binary conversions or hexadecimal to binary conversions. Having these concepts incorporated into the level could add another layer of difficulty for players to work through while also teaching them about core concepts that are covered in required computer science courses at CSU.

#### 4.6. Level Four

The fourth puzzle brings the player to a courtyard. This level is split up into three main parts. The optional first part is the Guide, the second is where they apply their knowledge of loops by practicing with examples, and the third is where they are pushed to think of loops more abstractly by being presented with a logic puzzle whose answer requires a robust understanding of how loops work.

Before describing each detail of the puzzle one by one, I believe it helps to differentiate features in the puzzle that are randomized versus features that are not. As Table 1 displays, features of the left are static features, while features on the right side required a random number generator in their implementation. Seeing these features side by side should give the reader a clear reference in how the puzzle is randomized and provides an overview on some other game design elements used in the Level Four that I do not delve into when talking about my design but that were used in the puzzle, such as collision boxes.

Non-Randomized Features	Randomized Features
Tile map	What inmate spawns in during level generation
Collision boxes for NPC bounds	Where inmate spawns on the outlined inmate spawn path
Collision box for door spawning bounds	How far the door spawns from inmate
Loop label	Item number request from inmate
Simple guide	Starting index for loop puzzle

*Table 1. Table displays features in Puzzle 4 that were randomized on the right and features that are static or are not programmed with logic that involves a random number generator on the left.*

The Guide is where the player can practice walking through how a for loop works step-by-step. The player gets the option to choose what parameter they would like to test with. As table 1 shows, this is a pre-programmed feature and will give one of the variety of answers given in Table 2. The decision to make the difference between the starting index and the ending index relatively small was intentional, as it allows the player to walk through the steps in a reasonable amount of time. Moreover, the steps of the loop repeat, so the player should be able to discern the pattern quickly without needing fifty iterations.

Combination	Result
< , ++	<code>for(int i = 0; i &lt; 3; i ++)</code>
≤ , ++	<code>for(int i = 0; i ≤ 3; i ++)</code>
> , --	<code>for(int i = 3; i &gt; 0; i --)</code>
≥ , --	<code>for(int i = 3; i ≥ 0; i --)</code>
< , -- or ≤ , -- or ≥ , -- or > , --	Suggests viable options to player

*Table 2. Shows all the different parameters the player can enter and the resulting for-loop they will get to walkthrough on step by step.*

Once the player feels confident in their for-loop knowledge, they start the second portion of the puzzle. Here, they encounter inmates that will tell them that a magical door that gives items. In order to get these items, however, the player must have knowledge of how to manipulate the parameters of a for loop. Because each time they open the magical door the current item index is set to a different position and the player has no way to reset it. This is the most complex portion of the level design, so I will explain each part one at a time starting with the inmates.

In this first half of the level, they will encounter inmates one at a time. There are three different types of inmates that can spawn. Each is differentiated by a sprite in the game, but also a different identification number, a different inmate key, unique dialogue, and a different magical door color and direction.

For instance, inmate\_A, who has the identification number 001 has a manner of asking things in a very verbose in a mystical way. They also have a purple door assigned to them and claim that their door is northwest of where they are. The decision to use the word northwest instead of in the upper right is intentional, as it matches with their over-the-top attitude and

mystical nature. Purple is also a color commonly associated with wizards and royalty which I feel fits their personality. As for the other inmates: inmate\_B is supposed to characterize a very straight forward and polite inmate, while inmate\_C is supposed to represent a scientific character who is to the point but in a technical way that is supposed to make the player really think. This characterization aided in the development of each of the inmates' dialogue and the color chosen for their door. Although, it is fair to say the color of each of the inmates' matching door is subjective.

The dialogue of each of the inmates is derived from a JSON file. When the JSON file is parsed in Godot, the information becomes a dictionary. Each of the inmates has a key labeled inmate\_A, inmate\_B, and inmate\_C. The dialogue for the inmates is split into several sections. Inmate\_beccon calls the player over to the inmate, inmate\_ask tells the player what to do, and door\_color and door\_location tell the player where they will find the magical door where the loop machine resides. This dialogue is used in the first half of the puzzle. Once the player enters the second half of the puzzle, each of the inmates' dialogue becomes structured with an inmate\_thanks, which tell the player that the inmates are thankful that they player got them the item they needed, and then a passcode\_hint which tells the player the next steps they need to accomplish in order to finish the level.

When an inmate is clicked on, the dialogue popup is instantiated and added as a child in the tree. To see what the what the dialogue popup looks like see Figure 8. NCPPopup's root node is a built-in Godot popup window, which has a built in accept button for the player to read the message and then close the popup window. The popup also has a built-in property to dynamically add a button. Using this feature, I manipulated one of the buttons to be the response of the player, while the other button was a "Got it" message that would close the popup window. Using a Godot's match statements, the appropriate dialogue is cycled through every time the player's dialogue button is clicked by incrementing a counter that matches to the next dialogue option.



*Figure 8. Showing the dialogue popup for inmates. This is an example of the last dialogue option.*



I could have also included the player's dialogue in the inmate dialogue JSON file, but I opted not to for two main reasons. Reason one is that the JSON file is built specifically with the inmates in mind and the inmate key used in the JSON file is used almost everywhere where the inmate is involved. I wanted to keep the same structure throughout and not have the player dialogue added on as an afterthought. Reason two is that the player dialogue is so short that it did not seem necessary to use the JSON file for it. Nevertheless, in designing the dialogue this way I got to see how one might use a JSON file and how easy it was to add and modify dialogue for the inmates on the fly. Another way I could have done this was having the dialogue structured as a dictionary from the beginning, but after using the JSON file I do prefer the readability of how it is structured.

When the player clicks on an inmate, the door for each inmate spawns. Once the player clicks on the door, they are presented with a loop puzzle that looks like the following image in Figure 9. The loop puzzle allows the player to experiment with setting the other parameters of the a for-loop given a random starting index. When the player enters a correct combination, a popup will show up telling them so. A popup will also tell them if they have entered incorrect information and give them a hint to lead them in the right direction. This hint will tell them the combination of parameters they can choose from to get the correct item index based on whether of not the item number is greater than the starting index.



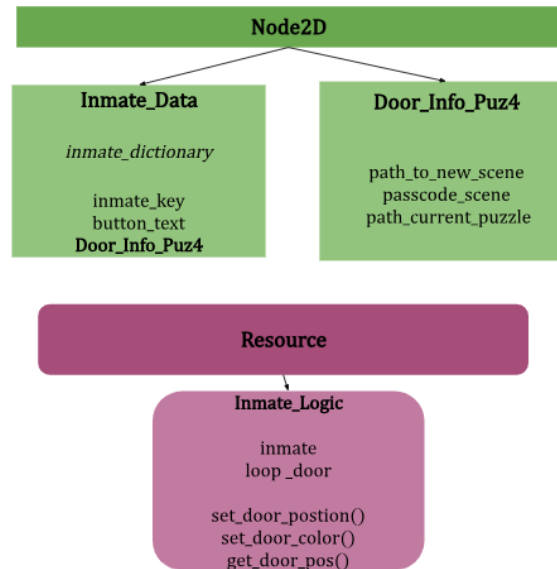
*Figure 9. Shows what a potential loop puzzle will look like to the player. Note that the drop downs give the player the option to select less than, greater than, less than or equal to, and greater than or equal to, as well as to increment each iteration or decrement each iteration.*

In the loop puzzle, the player is presented with the randomized item number that the inmate requested in their dialogue. The requested item for each inmate is derived from the Item\_Dict class. This is a lone script not attached directly to any nodes in a scene. It defines a static item dictionary. In this dictionary the keys are the inmates, and the value is of an



Item\_Num class which is another class whose purpose is to generate a random number. These two classes are set in the NPC script. The purpose of this dictionary is to be able to set a random item number for each inmate and then get that item number via the inmate key in the loop script.

In my research of using JSON files in Godot, I was also introduced to custom resources in Godot. I believe that there are clever ways these can be utilized in Godot, and looking back on how I utilized mine it would not be my preferred method here. My *inmate\_puzzle* resource was used more like a regular script than a resource. To get a visualization of the difference between using scripts that inherit from a Node rather than a resource refer to Figure 10.



*Figure 10. A visual to show the difference between the scripts utilized to store and set some of the inmate information. The Inmate\_Data and Door\_Info\_Puz4 class both extend the Node2D, while the Inmate\_Logic extends the Resource class of Godot. As explained in the text, this design is not recommended as it does not effectively utilize the strengths of a custom resource in Godot.*

The custom resource was initially supposed to be a place where I could set and change the position of both the inmate and the door matching to the inmate, but I decided the program would be better optimized if I just made it the set the door position randomly based on the position of the inmate and the door color based on the type of the inmate. It also has a function to get each door's position based on the key to each inmate. As hinted at previously, the door's position is derived by the inmate's global position.

The direction that each inmate says the loop door is always the same (northwest, left, or straight ahead), but the distance of the door from the inmate varies. For instance, inmate\_B always claims that the door is to the right of them, so the door's position is set to be the inmate\_B's current global position in the map subtracted by a two-dimensional vector that points to the right. The distance randomization is derived from a scalar multiplied from this subtracted

vector. This scalar range has two options depending on whether the door spawned within the range of the map or not. The first option is that the scalar is two times a random floating-point number between 1 and 1.5. The second option lessens this range so that the scalar is at most 2. This ensures that even if the inmate is right at the edge of the map that the door will still spawn in a reachable area. More details about the development for the door spawning is in the Verification section of this report.

If I were to redo this part of the project, I would opt for developing a simpler solution that involved instantiating the doors directly in the Puzzle 4 map, but this complicated way of implementing the door randomization pushed me to develop new solutions that I would not have thought of before. Another positive aspect of this implementation is that I have a more knowledgeable perspective on how one might go about designing level: a more straightforward way that gets the job done or in a more complex way that pushes me to approach the problem differently. Despite this method causing some trouble for me, I am glad I took the time to try a different approach as I believe it made me more knowledgeable on the intricacies of Godot game development.

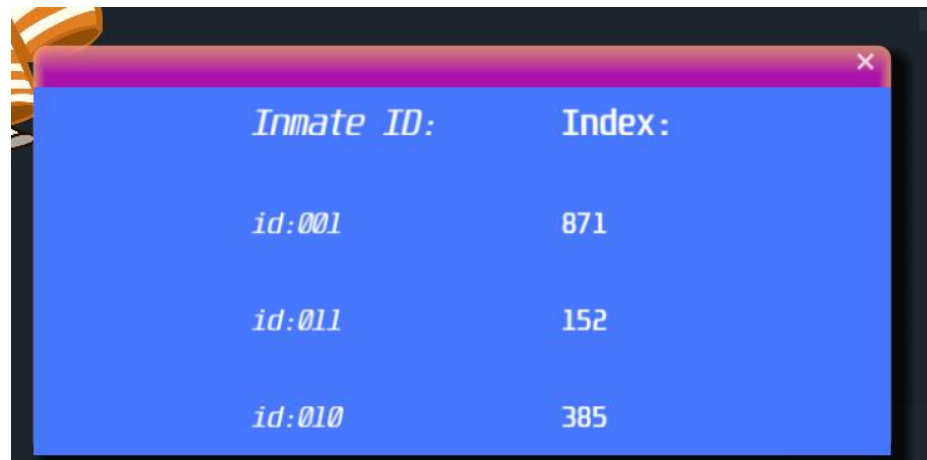
The spawning of each of the inmates is randomized by utilizing a Godot Path2D and an inherited PathFollow2D. A square path is outlined in the puzzle map and an inmate has a random chance of spawning anywhere on this square path. This method was taught in the Godot 2D Tutorial. The foundation of this method is based on the PathFollow2D's progress ratio attribute. This ratio is a floating-point number from 0 and 1 and determines where the current position in the path is. By randomizing this value, the position of where the inmate shows up is randomized. Because I want the inmates to stay in the main portion of the map, I added four bounding boxes to keep the inmates where they are supposed to be. As mentioned above, the inmates spawn one at a time in this portion of the puzzle. The inmate spawning order is random and kept track of in the Puzzle4Autoload. Once an inmate spawns and their loop puzzle is solved, they are removed from the list and another random inmate is chosen from the list. If an inmate spawns and their loops puzzle is not solved, they are added back to the list. Once the list of inmates is empty and all the loop puzzles have been solved, the last portion of the puzzle is initiated.

In this portion of the puzzle the player is greeted by all three inmates at once. This decision was made so that the player could re-access all the inmate's clues in the same iteration of the level. As mentioned above, this portion of the level gives the player a chance to think about loops more conceptually while also expanding on their previous knowledge in the first portions of the level. This time they are tasked with finding a potential starting index for the loop instead of manipulating the end index to fit their needs.

The second half of the puzzle takes more inspiration from logic puzzles one might find in an escape room. However, the idea was still to make what the player needed to do straightforward enough so they are not caught up on the directions and will use more of their time on the application portion. Each of the inmates and two extra popups give the player hints alluding to how to find the passcode.

The extra popup explains to the player that there is only one answer for the passcode. It also explains that the passcode changes based on how many times the inmates have entered the courtyard but have stopped working correctly since the player has entered. The inmates explain to the player that they know the door's passcode is the starting index when the player first entered the courtyard. This leads to the player needing an ending index to subtract from.

One of the inmates gives them this information, by explaining to the player that they need to add up the numbers in a popup to get the ending index. In this popup are two columns, one labeled Inmate ID and the other labeled Index. It is assumed that the player understands that they are only supposed to add up the numbers in the second column. Refer to Figure 11 for a visualization.



<i>Inmate ID:</i>	<i>Index:</i>
<i>id:001</i>	871
<i>id:011</i>	152
<i>id:010</i>	385

*Figure 11. Shows what the popup might look like for the player. Note that the indexes are based on what the player enters in each loop puzzle.*

I could have made the generation of these numbers random, but I instead use the user-entered ending index of each of the loop puzzles. I did this because I originally had a more complex idea on how to implement the second half of the puzzle that did not pan out. The generation of this popup is stored in the Puzzle4Autoload, which stores the ending index that the player enters as well as the ID of the inmate that asked for an item at that time. As a side note, the IDs of each inmate do not serve a purpose in the game. They were used during development before the inmate sprites were implemented to differentiate between inmates. Now they serve as a red herring of extra information similar to the design of an escape room puzzle.

Another inmate tells the player that there is a record of how many times they have looped the courtyard in the top upper left corner of the map. Here, there is label that updates according to a signal that is emitted each time the puzzle loads up again. The signal to update the loop counter is emitted in the puzzle 4 script's *ready()* function, and once the signal is emitted the Puzzle4Autoload provides the count.

The last inmate clarifies once again that there is only one passcode answer. But they have the potential to confuse the player because they say the player could either add or subtract one from the ending index, because the interworking of the passcode door only uses less than or

greater than signs. The idea is that the player understands that they need to subtract one from the sum, because they are attempting to get back to a lower starting point index. I will be the first to admit that this part of the puzzle is the most confusing and if I had more time, I would update it so that it is a lot more obvious to the player about what I am attempting to get them to understand.

Moving on, once the player sums the numbers in the popup, subtracts the number of times they have looped, and subtracts one; they obtain the passcode for the door. Once they enter this passcode, they are greeted by a screen that tells them that they have finished the game and have the option to go back to the menu.

Although I am proud of what I was able to accomplish given I had almost zero game development experience, this puzzle could have benefited greatly from a more robust planning process. I have a lot of features implemented that did not need to be as complicated as they are and took a lot of development time that could have been used to create a more enticing experience for the player, especially in the last portion of my level. If I had taken the time to think more about how I was going to implement the last portion, I think it would really improve the experience of the player and make them feel more comfortable working with for-loops.

## **5. Verification**

### **5.1 Testing Level One**

To test puzzle one, an implementation was made, and the game was then played as though a player were engaging with it. The newly added elements were thoroughly tested to avoid issues that would break the game and were confirmed to appear as intended. When the code reached a particular function, it was commonly used to print out on the console to ensure that it was operating correctly and using the desired functions in the right order. Because of an unfamiliarity with Godot and a preference to debug in a familiar manner, the debugger as much as it should have.

When problems arose, the implementation was divided into manageable steps, each of which was tested to guarantee functionality before moving on to the next. This method assisted in identifying any components that were not operating as planned. Resources like Google and the Godot forums were consulted for help if the specific issue could not be determined. Even though there might not have been a direct answer offered by these forums, they did direct the search within the code, which helped with independent problem-solving.

### **5.2 Testing Level Two**

The testing of the second level entails making sure all puzzles and player interactions function correctly. The puzzles should be generated with random variables that require players to solve each time. Players entering the correct answer should complete the puzzle and advance to

the next step. Player and robot interactions should also change based on the completion of the puzzles.

### 5.3 Testing Level Three

The testing of level three mainly consisted of running the level as a player. All the different elements of the puzzle were verified by acting as a player and solving the puzzles throughout the level to see what a player would typically experience.

Since this level consists of many pop-ups, both interactive and static, there were many runs of just ensuring that all the pop-ups worked correctly and appeared in a similar manner to maintain gameplay consistency. There was also a lot of effort into making sure that the interactive parts of certain pop-ups worked efficiently with other level nodes to verify that the correct pop-ups were only accessible after a certain point in the gameplay.

Additionally, randomly generated numbers also play a big part in completing this level. Random numbers are generated for both the bitwise operation and binary conversion puzzles with every new run of the game. It took a fair amount of time to make sure that the bitwise operation problems and binary numbers were displayed in a clear and obvious way. The same conditions were applied in making sure that the generated solutions to the puzzles were mathematically correct.

During the verification process of the level's functionality, several issues emerged. Notably, the bitwise operation puzzle would reset entirely if a player submitted an incorrect answer to the binary conversion puzzle or exited the scene prematurely. This setback occurred because upon returning to level three from the global scene where the door passcode is entered, level three would restart. To address this, a flag was housed in an autoloading script that tracks whether the player has successfully solved the bitwise operation puzzle. Once solved, the flag remains true for the duration of the game session, ensuring that players do not have to repeat unnecessary steps. Consequently, they retain access to the binary puzzle without having to solve the bitwise operation puzzle repeatedly. Other small issues and bugs were solved by changing the level's node structures and physical level format.

Level three testing verification resulted in a completed level that successfully challenges user to apply their skills in bitwise operation and binary conversion logic.

### 5.4 Testing Level Four

Throughout testing of level four, there was extensive use of the debugger and the console to test that things were working as intended. In this section I will walk through my process for debugging and testing three of the most essential parts of my puzzle.

The implementation and testing of the loop puzzle were a fairly simple process despite Godot's for-loops not having the same syntax of the classical for-loop that the level attempts to teach the player about. Once the player inputs the parameters in the for loop puzzle, a function called *calculate\_loop()* takes all these parameters and finds the correct case to match it. There are two levels of cases to check. The first case checks whether the randomly generated starting index is greater or less than the end index. The second case checks whether the inequality in use is the

regular inequality or the or equal inequality and whether increment or decrement has been selected. If none of the cases find a match, then the loop popup tells the player that they need to try again as the parameters did not match one of the suitable cases. Figure 12 portrays what will happen if the player attempts to enter incorrect parameters.



*Figure 12. Alert message shows up after player enters loop parameters that do not result in the desired item number.*

To give an example, if the starting index generated is greater than or equal to the user entered end index, then the program will then check what inequality applies— greater than or greater than or equal to, and make sure that the decrement option is being used before a loop result is calculated. Notice that it does not check if the inequality is less than or less than or equal or if the player chooses increment. This is because a for-loop structured with the starting index greater than the ending index using a less than sign would not run as the second condition would always be false. Furthermore, increasing the counter on each iteration of the loop with these example parameters would cause an infinite loop. To sum it up, these cases are in to ensure that the player does not enter parameters that would not allow the loop to run or cause an infinite loop.

One of the most cumbersome features to implement was the door spawning correctly. Through testing, a solution to the issue of the door spawning outside of the puzzle map became apparent. As mentioned previously in the Detailed Design section for Level 4, because each of the doors are spawned in during runtime and are abstracted through the custom resource and inmate class, I had trouble guaranteeing that the door would spawn when inside the bounds of the map because I did not yet understand how to I could access the door's position in a meaningful way.

My original idea was to use a collision polygon that outlined the boundaries of the map and have a signal emitted each time the collision polygon detected the door spawning. If the signal was not emitted, then the door's position was to be reset to with another random scalar. This idea did not pan out because as discovered through debugging the collision polygon does not detect the door spawning because it is abstracted through the inmate class.

To fix this issue, I added an additional `StaticBody` that would be recognized as colliding with the collision polygon in the Puzzle 4 scene. This proxy static body would be set to the position of the door and would dictate whether the door was in the correct bounds or not. If the case was that it was too far, the position of the door would be randomly reset with a lower randomization range. This, however, was not the end of my issues with the door spawning because I still was not getting the intended results. When the door was spawning out of bounds it was freezing the game.

I printed the position of the door and the proxy body and discovered that the position of the was changing far too quickly for the collision signal to be emitted. After some thinking, I found that if a timer was added for about 2 seconds, then the collision signal could finally be detected. This method has been tested extensively with twenty plus iterations, and I have not detected any more issues yet.

The last component of my puzzle that took a lot of testing was changing the passcode to be based on the information from previous parts in the puzzle. In our design, the puzzle resets the player's position each time they exit a new scene such as the loop user interface or the door passcode user interface. This fact makes it impossible to keep any data about what the user interacted with during runtime without using a global script that saves the information deemed meaningful. The passcode for the door is normally set to a random five-digit value each run-through, but now the mission was to get the passcode to be generated based on the previous user entered values. At first glance, this did not seem like it would be an issue as the data had been set in the `Puzzle4Autoload`. However, the function call to set the passcode came before the function call to change the passcode. This is because the puzzle 4's `ready()` is called before the functions that set up the second half of the puzzle occur. Walking through each function call in the debugger, a solution was developed where the function to set the passcode would have to await a signal to be emitted before the passcode would be set. This signal emitted once the necessary preparations to set the passcode had been completed. This method has been tested several times and has proved to be an effective solution in solving this issue.

## 6. Project Management

### 6.1. Team Qualification

Gretta Foxx is very familiar with how games are developed and designed. While she might be lacking in concrete game development experience, she has extensive programming experience and an innate ability to learn quickly. In the recent CPT Game Jam, she applied her previous general knowledge about game development and programming experience to construct a working game. Also, during the Game Jam, Gretta was tasked with creating the custom assets for the game. Her experience and genuine passion for creating these assets will improve the overall aesthetic of the game. She also has experience with music and knowledge of how to source game music, which will aid in creating an engaging and

replayable gaming experience. Lastly, Gretta prospered at her summer co-op with embedded systems projects. Similar to this project, she started with something unfamiliar and was able to successfully rise to the challenge at record speed. All in all, Gretta will not only serve as our artistic backbone for the project, but also set the pace of progress for the project.

Yaxin Gu also has minimal game development experience but has completed some JavaScript game tutorials in the past, displaying her curiosity about game development and eagerness to learn more. During the recent Game Jam with her teammates, Yaxin focused her efforts on designing background elements that the player can interact with through tile map collisions, which is something that the team will need for our project. She is currently working on studying Godot tutorials so she can assist the team in creating a functional game. Yaxin is powerful in her ability to stay absorbed in a project or goal until it is finished. Furthermore, she plays video games in her free time which will help the team in thinking about how players think so the game does not become unnecessarily complicated. In addition to her genuine interest in video games and game development, she also has a multitude of work experiences where she was able to advance her communication skills and showcase her ability to expertly support the team to accomplish any task.

Similar to Yaxin, Sarah Ogorzaly started working on this project with very little game design experience, but she and her teammates recently attended the CPT-hosted Game Jam, where she got to aid in developing her first ever game. Through this experience, she was able to discern possible issues that arise when developing a game and understand the fundamentals of utilizing Nodes in Godot. In conjunction with this meaningful Game Jam experience, she also has a real passion for computer science concepts and spends a lot of time thinking about developing the best solutions. As her resume shows in Appendix A, she has previous work experience in retail displaying her communication skills. She was also a research assistant last summer, showing her willingness to delve deeper into subjects that require more time to digest. Sarah's comprehensive studies over these last three years and her desire to tackle new challenges make her a valuable team member.

Sidney Zweifel originally started her college career in filmmaking and production management, which makes her an excellent fit to be our team leader as it has honed her skills in group coordination and management. It has also taught her to be adaptable to how people work in the creative process. Despite not having traditional game development experience with a computer game, Sidney is a current employee in the Dan T Moore Makerspace where she works for CSU in Virtual Reality (VR) development. This VR development is performed in Unity, which is another popular game engine; meaning she has a concrete grasp on the fundamentals of game development organization and object hierarchy. Moreover, Sidney was an Engineering Peer Teacher (EPT) for CIS 151, allowing her to reinforce the most important underlying programming concepts with novice students. This experience showed her what students struggle with the most, which will be useful in thinking about what kind of puzzles would be of most benefit to them. Last, but certainly not least, Sidney completed a software



development internship over the summer where she successfully combined her exponential communication and outstanding technical skills. Her proven achievement as a team leader, software engineer, and her familiarity with Unity make her an ideal candidate to lead our team in completing a successful project.

As a final note, the team has successfully worked together all semester and has grown more comfortable with each other in these last months. This should allow for a smoother workflow as we are all aware of each other's work management styles and can pivot off of our individual strengths as team members and as computer scientists.

## 6.2. Task Assignment

The tasks and subtasks of each team member are as follows. Please note that some tasks have been added during development and that tasks denoted with asterisk have been modified during development:

Gretta Foxx

- Group's primary brainstormer and concept design artist
  - Walking us through her thought process
  - Constructing game design sketches so that group members can visualize ideas
- Game development
  - Partially in charge of ensuring smooth transitions between different scenes of the game
- Documentation
  - Summarizing basic Godot concepts
  - Explaining development of their portion of the levels
- Development of Level 1
  - Print Statements / Variables
- Assets
  - Created all the animations for the Player
  - Created all the animations for the Robot Enemies in Level 2
  - Created the computer asset in Level 1
    - Including all the folder icons and menu images
  - Created the end screen image

Yaxin Gu

- Group's game development and gameplay researcher
- Game development
  - Partially in charge of ensuring smooth transitions between different scenes of the game
- Documentation

- In charge of explaining our design process in detail, which will include explanations on how we structured everything together and why we chose to go in certain directions during development. Will show others different paths they can take if they decide to add on or modify our game
- Explaining development of their portion of the levels
- Development of Level 2
  - If-Else

#### Sarah Ogorzaly

- Group's background and environment designer\*
  - \*Allocation of this task changed during development: Each group used royalty free assets to design their own backgrounds
- Randomization tools
  - Partially in charge of creating tools with scripting languages to ensure levels are replayable through randomization\*
  - \*This changed during development: Although JSON was used in the game as a way to store data that could be randomized, randomization was programmed by using Godot's in game random number functions. Because each group member is responsible for their own level the logic for randomization was developed for each individual level. The caveat to this is the passcode generation for the door that transitions between levels. For every level, except level four, the passcode is generated randomly through a global script.
- Scene Manager
  - Partially in charge of researching and developing a scene manager that allows for seamless scene switching to move between levels
- Documentation
  - In charge of creating a very basic tutorial so others can get started creating their own level in our game
  - Explaining development for their portion of the levels
- Development of Level 4
  - Loops
- Assets
  - Created all the inmate assets
  - Door asset used in Level 4

#### Sidney Zweifel

- Team management and coordination
  - Facilitating weekly group meetings
  - Keeping team members updated on group discussions and everyone's personal duties

- Visual brainstormer
  - Creating engaging demonstrations and presentations to demonstrate her thought process
- Randomization tools
  - Partially in charge of creating tools with scripting languages to ensure levels are replayable through randomization
- Scene Manager
  - Partially in charge of researching and developing a scene manager that allows for seamless scene switching to move between levels
- Documentation
  - Organization of the documentation
  - Explaining development for their portion of the levels
- Development of Level 3
  - Bitwise operation (&, |) and binary-decimal conversion
- Assets
  - Created digital art used for pop-ups in level 3

#### Report Sections:

- Gretta Foxx
  - 3.3 Defining Technical Specifications
  - 3.4 Enumerating Design Concepts
  - 4.3. Level One
  - 5.1 Testing Level One
  - 6.4 Deliverables
  - 8. Conclusion
- Yaxin Gu
  - 3.1 Identifying the Unmet Needs
  - 3.2 Determining the Design Constraints
  - 3.5 Selecting Design Concepts
  - 3.6 Standards Compliance
  - 4.4. Level Two
  - 5.2. Testing Level Two
- Sarah Ogorzaly
  - 1. Statement of Problem and Background
  - 4.1. Scene Manager
  - 4.2. Global Randomization
  - 4.6. Level Four
  - 5.4. Testing Level Four

[illegible]

Figure 13. Fall 2023 Gantt Chart

See [Fall 2023 and Spring 2024 Gantt Chart](#) for a more detailed view of our timeline.

During our weekly meetings, each team member would showcase what they had been working on throughout the week and also go through any struggles that they were facing. This not only allowed us to have a more collaborative environment, but also allowed every team member to get a feel for how each level was progressing. Each team member had their own branch on our project's GitHub and pushed their updated code to that branch with comments, which other team members were able to use as reference. In addition to our game, we have a reference document that hosts a description and links to all the resources that have been used in our game. Also, we will provide comprehensive documentation that details how to create a new level in our game, which can be used as a guide for anyone who may want to add to our game. For our last deliverable, all the team members' contributions have been successfully combined into a single game file, resulting in a completely functional and polished game that is interesting, fun, and instructive.

While we planned to make a fair amount of game assets on our own using digital art software, we also wanted the option to purchase game assets via Godot marketplace. The plan was to purchase game assets for the artistic and musical aspects of the game as we will not have time to create everything from scratch. However, royalty-free assets were prioritized throughout the development process. The budget table below overestimates the funds necessary for our game in preparation for any last-minute changes we could possibly have made to our project.

Item	Supplier	Quantity	Unit Price	Total
Game assets	Godot	1	~\$150	~\$150
			<b>Total</b>	<b>~\$150</b>

Table 3. Budget list

Despite allocating \$150 to possibly spend on game assets, we were able to construct our game using open-source assets or creating our own art for front-end aspects of the game.

### 6.6 Communication and Coordination with Sponsor

Since our project is student-proposed, we do not have a sponsor to coordinate with. However, we are in contact with our advisor, Professor Robert Fiske, in person and via email/Discord.

For the Fall 2023 semester, we only met with Professor Fiske in person when necessary. This was due to our project still being in the early development stages as we were still learning how to use Godot and have only made the most progress in our research phase. Additionally, Professor Fiske's advising style is tailored to how we want our timeline to play out. While he is always available for meetings and responds quickly to questions, we have found that it is not always necessary to meet when we do not have specific questions or game development updates to be addressed.

By abiding by our plan described in our Fall 2023 Gantt Chart above, we accomplished all that we planned to do during that semester since we completed a multitude of research goals surrounding game development and Godot specifically. Even more importantly, we completed several simple games to get comfortable with Godot. Through all of this, Professor Fiske was there to answer any questions we had and to guide us in the best direction.

For the Spring 2024 semester, we met every week to discuss game development, debugging, and other senior design (EEC 494) related updates. Each member demonstrated what they had worked on in the week preceding a meeting. Professor Fiske gave us feedback on what we were doing effectively and aspects that we could improve upon or add to the project. As for testing a prototype of our game, we as a team play tested the game after every major code merge of the project before presenting our progress to Professor Fiske.

Additionally, we regularly reached out to our advisor via Discord or email for any questions regarding game development and coursework relating to Senior Design.

## 7. Professional Awareness

Even though we are still novices in our careers as software engineers, we have been continuously reminded to uphold our professional and ethical responsibilities throughout our time at college. This not only means we will take the utmost effort in sourcing our inspirations and academic references, but also means we will take the time to educate ourselves on the current issues concerning game development. In our efforts to contextualize educational game development we not only develop a more nuanced view of the subject, but we also become aware of the most ethical ways we can develop educational games without crossing moral boundaries.

With that being said, we would like to take the time to mention our primary sources of inspiration for this project. There already exists a sleuth of computer science and programming games for the public. Some of the most prevalent include Human Resource Machine [8], Code Monkey [9], and Scratch [10]. Note that these were sources of inspiration before development.

In Human Resource Machine, players control workers by reorganizing blocks of code to automate their behavior. Its wacky premise poking fun at corporate culture keeps players engaged. As a team, we tested this game and took note of how it slowly introduced new programming and logical concepts by relying more upon intuition from the player than strict and verbose directions. We also enjoyed that despite the game teaching the player basic assembly code, it has a lot of personality with its writing which frequently includes quips from characters. We hope our game is fun in some of the same ways.

Code Monkey is the most like the game we planned to develop in that a major goal of the game will be teaching players. It is owned by a company of the same name that specializes in making programming games that teach K-8 students. Games at a lower-level range are block-based, intermediate games are text-based, and their more advanced courses focus on the creation of games and even programming their own chatbot in Python. What we are most inspired by with Code Monkey is its showcase of starting from knowing no programming to developing your confidence as a programmer and tackling your own meaningful projects. Reviewing what we were able to accomplish in our game, this was a difficult feat to achieve. Some of the puzzles are more beginner friendly than others, but with time and determination from the player we believe that the challenges in each puzzle are achievable.

Scratch is an MIT software project that allows users to create their own games. It uses a block-based programming method where users customize the look of their game and control all aspects of the game development including sensing player movement with user input to event planning to programming the motion of non-playable characters. Scratch is a software that many future software engineers and game developers begin their journey with, so it serves as a role model for our project and projects like it can inspire. In the end we did not

take a lot of game elements from Scratch, but it did aid us in guiding our philosophy in making something that serves to motivate others to make something that challenges them academically and creatively.

Please be aware that these are just some primary sources we have considered; it does not serve as an exhaustive list of all our inspirations. As we moved further into developing the game, we were sure to note all the tutorials and references we used. This is also the time to mention that any outside assets we use for our game are cited in our documentation. Moreover, credit for any group members who take the time to make their own assets for the game is also listed in the Task Assignment section.

We must also acknowledge the responsibilities we have in our creation of an educational game. Because our project is meant to teach people, it is our duty to give people the most accurate information we can. Although this is easier said than done, it will give us a clean conscience in that we tried our best to create something that students could learn from. Moreover, because other primary goals of the project are to allow others to use our project to learn about game development and inspire them to create their own levels that teach computer science subjects that they have an interest in; it is also our professional responsibility to take great care in developing the documentation for our project so that others do not struggle to follow our thought process and can develop new ways which work for them. However, it should be noted that our documentation does not serve as a single point of entry for developing games, rather it will be another helpful hands-on resource. Because this is such a formidable undertaking, we must note that we have limited the project to creating the most polished end product we can. Although it disappoints us not to get to implement all our ideas, it will allow us to tie all loose ends and focus on creating the best experience we can.

This project, like any other project, also showcases our pursuit of lifelong learning. We have learned a lot during the development of this project and that these skills should prove valuable in our careers to come. Although game development might seem like the only relevant technical skill we learned through the project, we also showcased our ability to take on unfamiliar challenges enhancing our technical and communication skills.

## **8. Conclusion**

Computer science is a vast and intimidating topic, which requires abstract thinking and excellent problem-solving skills. To provide learners with a focused, fun, and engaging learning experience, we have used the medium of an educational video game. Our game eases players in by starting with very basic ideas and progressing to more complex topics, and incorporates replayability through randomization, which allows players to retry levels as many times as they need to feel proficient in each topic. It is our hope that players can practice fundamental computer science subjects with our game but is our overall wish that our game serves as



inspiration for others to create their own games that help them study their academic concepts in a unique way but valuable way.

Our team is proud that we were able to successfully create a game that enhanced our technical, creative, and teamwork skills, but like any project it is important to reflect on the process and identify where improvements can be made. Two of our main pitfalls during game development were productivity and performance.

Our productivity may have increased if we prioritized developing reusable assets together. For instance, the door asset that implements our scene manager in the game would have benefited from being developed by all members of the team, so that everyone knew how it worked and could more easily manipulated it to everyone's needs. We also could have spent more time learning about how to use GitHub, so we did not run into so many merge conflicts when we combined our work.

The game's performance is also something that can be improved upon. Since we had little experience developing a game, most game elements were implemented in a way which favored results over efficiency. As a result of this, some levels take a while to load relatively simple elements. Moreover, there was extensive use of Autoloads in our puzzles, which weigh down game performance. Perhaps if we had more time, we could develop a more elegant solution to creating our puzzles that are not heavily reliant on global scripts.

Possible future work for our project includes creation of levels on more complex concepts computer science concepts such as sorting algorithms, creating levels where the whole group collaborates to create a more intricately developed puzzle, and going back to redefine our game in how it is organized and in how efficient it runs.

Our experience in creating an educational video game to teach computer science concepts has been both rewarding and enlightening. We aimed to provide learners with an engaging and immersive experience, easing them into complex topics while creating an enjoyable learning experience. Although we celebrate our accomplishments, we acknowledge areas for improvement, particularly in productivity and performance. By reflecting on our process, we recognize the importance of collaboration and efficient implementation. Moving forward, we envision expanding our game to encompass more intricate concepts, fostering greater collaboration, and refining its efficiency. Our hope is not only to empower learners but also to inspire others to explore innovative ways of studying academic concepts.

## 9. References

- [1] W. Toh and D. Kirschner, “Self-directed learning in video games, affordances and pedagogical implications for teaching and learning,” *Computers & Education*, vol. 154, p. 103912, Sep. 2020, doi: 10.1016/j.compedu.2020.103912.
- [2] J.P. Gee, “Learning and Games”, *Learning and Games*, pp. 22-40, 2008, doi: 10.1162/dmal.9780262693646.021
- [3] C. Steinkuehler and K. Squire, “Video Games and Learning,” in *The Cambridge Handbook of the Learning Sciences*, 3rd ed., R. K. Sawyer, Ed., in *Cambridge Handbooks in Psychology*. , Cambridge: Cambridge University Press, 2022, pp. 281–300. doi: 10.1017/9781108888295.018.
- [4] Ascalon, Anecia, “Escape Rooms: Everything You Need To Know (2023)”, <https://theescapegame.com/blog/what-is-an-escape-room/> (United States: The Escape Game, 26 October 2021)
- [5] Cleveland State University, “Bachelor of Science in Computer Science”, <https://engineering.csuohio.edu/eecs/bachelor-science-in-computer-science> (Cleveland, OH: Cleveland State University, 2023)
- [6] Persson, Jonathan and Hammar Nicolas, “Designing for Replayability: Designing a game with a simple gameplay loop for the purpose of being replayable”, <https://uu.diva-portal.org/smash/get/diva2:1672135/FULLTEXT01.pdf> (Uppsala, Sweden: Uppsala University, 18 June 2022)
- [7] "ISO/IEC 25010." <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>
- [8] “Tomorrow Corporation : Human Resource Machine.” Accessed: Nov. 20, 2023. [Online]. Available: <https://tomorrowcorporation.com/humanresourcemachine>
- [9] “Coding for Kids | Game-Based Programming | CodeMonkey.” Accessed: Nov. 20, 2023. [Online]. Available: <https://www.codemonkey.com/>
- [10] “Scratch - Imagine, Program, Share.” Accessed: Nov. 20, 2023. [Online]. Available: <https://scratch.mit.edu/>
- [11] “How to ‘Change Scenes’ (without losing your player) | Godot 3.x Tutorial,” [www.youtube.com. https://www.youtube.com/watch?v=In\\_HYViDstE&list=LL&index=18](https://www.youtube.com/watch?v=In_HYViDstE&list=LL&index=18) (accessed Apr. 15, 2024).
- [12] “This Godot 4 Scene Manager Does it ALL,” [www.youtube.com. https://www.youtube.com/watch?v=2uYaoQj\\_6o0](https://www.youtube.com/watch?v=2uYaoQj_6o0) (accessed Apr. 15, 2024).

## Appendix A: Résumés of Team Members

### Gretta Foxx

[g.foxx@vikes.csuohio.edu](mailto:g.foxx@vikes.csuohio.edu) ♦ (440) 622-2034 ♦ Mentor, OH

#### EDUCATION

##### Cleveland State University

January 2021 to Present

Cleveland, OH

- Bachelor's in Computer Science expected May 2024
- Minor in Mathematics Completed Spring 2023
- Overall Grade Point Average: 3.94/4.0
- Dean's List: Spring 2021, Spring 2022, Fall 2022, Spring 2023
- Choose Ohio First Scholarship: Fall 2021 - Present

##### Lakeland Community College

August 2019 – May 2020

Kirtland, OH

- Associates of Science Degree, December 2019
- Overall Grade Point Average: 3.6/4.0

##### Mentor High School

August 2014 – May 2018

Mentor, OH

- Graduated Summa Cum Laude
- Enrolled in College Credit Plus Junior and Senior Year

#### WORK EXPERIENCE

##### Darko

May 2023 – August 2023

Bedford Heights, OH

##### Software Engineer Co-op

- Used communication skills to collaborate with team members effectively on projects
- Used time management skills to ensure that projects assigned to me were complemented in a timely manner
- Created excellent documentation on all software projects I worked on to ensure projects could progress smoothly when my Co-Op rotation was completed
- Developed custom documentation on how to implement new software in the company that will be used to train new hires for a smooth transition

##### DoorDash

July 2022 – May 2023

Mentor, OH

##### Delivery Driver

- Supported customer satisfaction by communicating clearly and professionally to deliver orders accurately and on time
- Used critical thinking to plan optimal routes to minimize delays and ensure orders are delivered on time
- Coordinated with multiple restaurant employees to ensure customer orders were correct and received in a timely manner

##### Pizza Hut

July 2019 – June 2021

Mentor, OH

##### Production Worker

- Coordinated with crew members to ensure food was prepared in a timely manner during hectic, fast-paced operations
- Developed skills in time management
- Communicated Clearly with Coworkers and Management
- Trained new employees to become educated and efficient at tasks
- Took initiative to find extra tasks when scheduled duties were complete

##### Lake Humane Society

August 2016 – August 2018

Mentor, OH

##### Volunteer

- Was responsible for the health and safety of rescue animals
- Helped potential adopters through the process of adoption

#### SKILLS & INTERESTS

- **Skills:** Java, JavaScript, C, C++, Godot, Linux, VS Code, HTML, CSS, SQL, Git and Python, Teamwork, Communication, Organization, Time Management, Problem Solving
  - **Interests:** Reading Novels, Playing the Guitar, Drawing, Traveling, Learning New Skills
- References available upon request

# Yaxin Gu

---

y.gu17@vikes.csuohio.edu | (216) 352-8391 | Cleveland, OH

## Objective

To work in a computer related field. To be able to contribute to the working field and to gain experience.

## Education

### **CLEVELAND STATE UNIVERSITY | JAN 2021 - PRESENT**

Cleveland, Ohio

- Major: Bachelor of Science in Computer Science
- Grade Point Average: 3.32

## Technical Skills

### **PROGRAMMING**

- Java, C++, C#, Python, HTML, JavaScript, SQL, Prolog

### **SOFTWARE**

- Microsoft 365, AutoCAD, SolidWorks, Blender, Godot

## Experience

### **COMPUTER LAB MONITOR | CLEVELAND STATE UNIVERSITY | JUN 2022-PRESENT**

Euclid, Ohio

- Help lab visitors with their needs
- Maintain and update lab equipment

### **RESEARCH STUDENT | CLEVELAND CLINIC | SEP 2022-NOV 2022**

Cleveland, Ohio

- Designed databases using SQL
- Used databases to organize clinical data related to genetic epilepsy

### **WAREHOUSE ASSOCIATE | AMAZON | JUN 2019-MAY 2020**

North Randall, Ohio

- Receive and sort inventory
- Get customer orders ready for delivery

### **RETAIL CASHIER | CAM INTERNATIONAL MARKET | JAN 2018-JUN 2019**

North Randall, Ohio

- Process purchases and returns
- Help customers with their needs

## Sarah Ogorzaly

s.e.ogorzaly@vikes.csuohio.edu (440)-539-3795

### Objective

Undergraduate computer science student seeking a software engineering position that allows me to utilize my creative problem-solving skills and technical expertise to make a tangible real-world impact.

### Education

- Cleveland State University, Cleveland, Ohio**  
 Bachelor of Science in Computer Science *Expected May 2024*  
 Grade Point Average: 3.97 / 4.0  
 President's List: Spring 2023, Spring 2022, Fall 2021  
 Dean's List: Fall 2022, Spring 2021, Fall 2020  
 Relevant Coursework: Machine Learning, Artificial Intelligence, Software Engineering, Operating Systems, Database Concepts, Computer Networks, Computer Architecture, Systems Programming, Language Processors, Data Structures & Algorithms, Comparative Programming Languages, Quantum Computing, Linear Algebra, Discrete Mathematics, Engineering Statistics, Technical Communication

### Skills

- |                          |         |              |                      |
|--------------------------|---------|--------------|----------------------|
| • Microsoft Office Suite | • C     | • JavaScript | • CSS                |
| • SQL                    | • HTML  | • Linux      | • Visual Studio Code |
| • Python                 | • React | • Java       | • GitHub             |

### Work Experience

Cashier | Lowe's Home Improvement | Brooklyn, OH *Summer 2022*

- Delivered excellent customer service and ensured that customer's needs were met in a professional and friendly manner
- Coordinated with team members to guarantee seamless store operations and fulfill essential front-end obligations
- Enforced safety and scam protocols by diligently following all detailed procedures

### Research Experience

Research Assistant | Cleveland State University | Cleveland, OH *Summer 2023*

- Conducted research on underwater swarm robotics, with a focus on the fundamental principles of underwater robotics and underwater Internet of Things
- Analyzed state-of-the-art research on swarm robotics algorithms including multi-agent reinforcement techniques, swarm intelligence algorithms, and heuristic algorithms to make an educated decision on the most effective strategies
- Actively engaged in constructive discussions with my advisor by preparing detailed weekly notes and bimonthly PowerPoints that summarized what I learned and how I could synthesize that knowledge to develop a solution for accurate, sustainable, and efficient water monitoring with underwater robotics

### Achievements

- Cashier of the Month at Lowe's Home Improvement based on Customer Satisfaction Reviews *July 2022*
- Washkewicz College of Engineering Student Achievement Award for Computer Science Spring 2022 at the Sophomore Level at Cleveland State University *Spring 2022*

# Sidney Zweifel

Greater Cleveland  
Incoming Software Developer at The J.M. Smucker Company

Phone: (330)-421-5807 Email: [sidneyzweifel1@gmail.com](mailto:sidneyzweifel1@gmail.com) LinkedIn: <https://www.linkedin.com/in/sidneyzweifel/>

Versatile worker and goal-oriented Computer Science student with a background in peer teaching, VR development, and software development. I enjoy adapting to constantly changing conditions at work and in life. I have a passion for tech and entertainment, thus I aspire to go into a career that integrates both art and STEM.

## EDUCATION

### Bachelor of Science in Computer Science || GPA: 3.6

Minor in International Film

Cleveland State University | 2021 - 2024 | Cleveland, OH

### High School Honors Diploma

Cloverleaf High School | 2014 - 2018 | Lodi, OH

## WORK EXPERIENCE

### Software Developer Intern

The J.M. Smucker Co. - Information Services | May 2023 - Aug. 2023 | Orville, OH

- Used PL/SQL and Oracle APEX to build applications for material suppliers to modify database information
- Utilized BASH scripting to implement improvements in a supply chain workload automation application

### Virtual Reality Developer

Cleveland State University - Washkewicz College of Engineering | Nov. 2022 - Present | Cleveland, OH

- Utilize Unity, C#, and MiddleVR to develop VR applications for CSU's MakerSpace
- Implement improvements to the VisCube virtual training program

### Engineering Peer Teacher

Cleveland State University - Washkewicz College of Engineering | Jan. 2022 - Dec. 2022 | Cleveland, OH

- Assisted with questions and Java programming projects for a class of 70 students
- Planned for and held weekly review sessions for CIS 151: Invitation to Computing

## ACHIEVEMENTS

### Margaret Taber Scholarship

Cleveland State University | Aug. 2021 - Present | Cleveland, OH

Received through Washkewicz College of Engineering

### Jack, Joseph, and Morton Mandel Honors College Grant

Cleveland State University | Apr. 2020 - Present | Cleveland, OH

Scholarship received through academic excellence

### Dean's/President's List

Cleveland State University | Aug. 2018 - Present | Cleveland, OH

Maintaining a cumulative GPA of at least 3.25/3.90

## SKILLS

**Languages:** Java, C, C++, C#, Python, SQL, PL/SQL, Javascript, HTML/CSS, BASH, GDScript

**Frameworks/Technologies:** Linux, React.js, Unity, Godot, Oracle APEX, MiddleVR, Flask, Git/GitHub

**Other:** teamwork, adaptability, time management, public speaking, OOP, software/VR development