Software Requirements Specification (SRS)
Leveled Logic

Team: Jonathan Kang, Connor Klein, Gabriel Shahrouzi, Eric Ta, Nathan Wright
Authors: Jonathan Kang, Connor Klein, Gabriel Shahrouzi, Eric Ta, Nathan Wright
Customer: Middle School Education System
Instructor: James Daly

1 Introduction

- (Section 1) Introduction
    - Identifies the purpose and scope of Leveled Logic.
    - Provides definitions, acronyms, abbreviations, and organization for Leveled Logic.
- (Section 2) Overall Description
    - Gives a brief introduction of what information is covered in this section.
- (Section 3) Specific Requirements
    - Gives an enumerated list of requirements which uses a hierarchical numbering scheme for Leveled Logic.
- (Section 4) Modeling Requirements
    - Specifies the bridge between the application domain and the machine domain.
    - Describes the use and notation for each use case diagram.
- (Section 5) Prototypes
    - Describes what the prototype shows in terms of system functionality and how to run the prototype.
- (Section 6) References
    - Provides a list of all documents referenced in the SRS by identifying each document by title, report number, date, and publishing organization.
    - Specifies the sources from which the references can be obtained (includes the project website).
- (Section 7) Point of Contact
    - Provides the contact information of the professor.

1.1 Purpose

The SRS document organizes various information about the Leveled Logic game including modeling requirements and prototypes. The idea is to provide students that are in elementary school or early middle school an educational experience involving logic gates and wires. These ideas are highly abstract which the game tries to break down by providing various levels for students to play.

1.2 Scope

The scope of Leveled Logic is to make an educational game which is both fun and an effective tool to learn about logic gates. Detailed information about the game including modeling requirements and prototypes are included on the Leveled Logic website. The application of Leveled Logic is to be used as a game for primary and middle school students. The benefits include fulfilling certain educational standards such as DESE and teaching them complex problem solving skills. Leveled Logic provides an interface to interact with the levels in the game. This interface provides various features such as to pause and select a particular level. In each level, students are able to spawn logic gates and wires to connect to each other. Furthermore, the levels are numbered starting from one to indicate the level of easiness. For instance, the first couple of levels are tutorials which allow students to gain an understanding of the controls and how to navigate the game.

1.3 Definitions, acronyms, and abbreviations

- Leveled Logic
    - Leveled Logic Game
- Logic Gate
    - is a device that acts as a building block for digital circuits. They perform basic logical functions that are fundamental to digital circuits.
- Wire
    - connects two different logic gates together carrying the output signal from one to the input signal of the other.
- Truth Table
    - Shows how the truth or falsity of a compound statement depends on the truth or falsity of the simple statements from which it's constructed [6]
- Level
    - An instance of the game that involves solving a task of putting together wires and gates.
- Player
    - The student playing the game.
- Godot
    - The game engine used to design Leveled Logic.
- DESE Curriculum DLSC Framework
    - The educational framework for primary school students.

1.4 Organization

The SRS document after the introduction delves into the details about Leveled Logic. This includes various models and game specific details such as how to run the game and interact with it. Section 2 provides an overall description, product perspective, product function, user characteristics, constraints, assumptions and dependencies, and apportioning of requirements. Section 3 provides details of specific requirements. Section 4 provides modeling requirements which includes a data table for each one. Section 5 includes the prototype for Leveled Logic, describes its functionality, how to run it, and provides sample scenarios. Section 6 provides references to materials used including a reference to the Leveled Logic website. Section 7 is the point of contact with the person who oversaw this project (ie. grading), namely the professor.

2 Overall Description

This section provides the product perspective, which describes the context for the product. Product functions appear after which summarizes the major functions that Leveled Logic performs. User characteristics provide expectations about the user (e.g., background, skill level, general expertise). Constraints give English descriptions of safety-critical properties and other properties if violated, the system will not perform properly. Assumptions and dependencies provide assumptions made about the hardware, software, environment, and user interactions. Apportioning of Requirements which are based on negotiations with customers provide requirements that are determined to be beyond the scope of Leveled Logic but may be addressed in the future.

2.1 Product Perspective

Leveled Logic is a game designed for primary and middle school students where they navigate levels designed to test one's ability to use logic gates. The functionality for the game is accessed through user interface which provides access to the main menu and an option to select a level. On a level, a player can move around and place and remove gates and wires. A level is successfully completed by the user if the user's solution matches the expected output from the level. These inputs and outputs are determined by truth tables. This game is a standalone game whose sprites have been developed by Johnathan Kang and made using Godot.

The interface includes various menus such as when a user first loads the game and after the user completes a level. The main menu includes a level select menu, a how to play menu, a credits menu, and an exit button. The how to play menu provides details of how to play the game including key controls. The credits menu is to acknowledge certain resources and developers who contributed to making the game. For instance, Jonathan Kang is attributed for making the sprites used in Leveled Logic. The level select menu allows the user to select a level or go back to the main menu screen. The pause screen will be available to users when they are inside of a level and they choose to stop the game. The pause menu shows multiple options including resume the game, restart the level, or go to the main menu. The level solved menu provides the user the choice to go to the next level, restart the level, level select menu, or go to the main menu.

The hardware interface is not necessary for our game because there is no hardware required besides a standard keyboard and mouse that is usually accompanied with a standard personal computer. As a result, there are no hardware limitations for our project. Our software interface relies on the programming languages GDScript and the game engine Godot. The operating systems it runs on are Windows and Linux. Since there are no communication

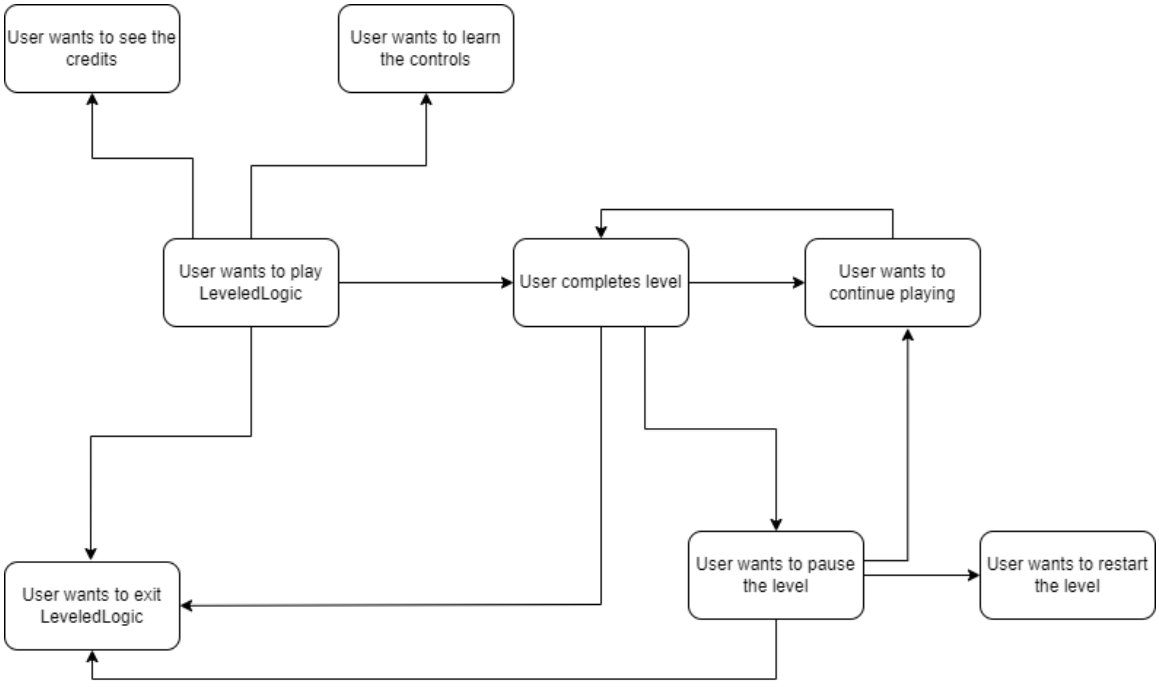
interfaces, the project is not limited by them. There are no other constraints that we are aware of that would restrict our project in a noticeable way.

The RAM required to play the game is what is a small percentage of what is normally found on that of a personal computer, namely 8 GB of RAM. The storage required to play Leveled Logic should be at least 10 GB. The operations required by Leveled Logic have been described to great detail in the paragraphs before so they will not be iterated over again. Site adaptation operations only apply to changing the game based on the operating system. Since the game needs to operate on both Windows and Linux, the containerization for the Leveled Logic needs to be modified slightly to work on both operating systems. Leveled Logic however does not need to be modified.

2.2 Product Functions

Leveled Logic allows users to interact with the UI to select a level. Within the level, the player will spawn logic gates and wires in order to pass the level. Passing the level requires matching the outputs with the expected outputs of that level. The user is freely allowed to switch levels or view the controls necessary to play the game at any moment - either from the main menu or while playing a level. Additional information that is included is viewing the credits or in other words contributors that helped make the game.

Figure 2.2.1 – Goal Diagram



2.3 User Characteristics

Users are expected to be in middle school or higher education. The users should have a familiarity with computers and how they work from a high level. Users are expected to be familiar with video game mechanics such as navigating menus, changing options, and movement.

2.4 Constraints

Leveled Logic has constraints focused on the system the game runs. The game requires a modern windows edition or an up to date linux installation. An up to date web browser and internet connection is needed to download and update the software. The game does not have any security restrictions because it does not collect any information.

2.5 Assumptions and Dependencies

Leveled Logic uses a mouse and keyboard to navigate the interface. The hardware required is the bare minimum that is present on most modern personal computers. The software on the computer needs to be able to run an executable exported from Godot. The game runs either on the Windows or Linux operating system.

2.6 Apportioning of Requirements

Some requirements are beyond the scope of the project due to time and resources. These requirements include a story, the ability to share level solutions, a timed mode or challenge levels, and the ability to save progression.

A story for the game will not be included in the game. A story would add to the functionality of the game by making it more engaging for the player. This was identified as being too challenging and unnecessary for our topic of the game. A story would also take away from the educational aspect of the game and the ability to play any level without worrying about missing out on the story.

The ability to share level progression is for peers to collaborate with each other and explain their reasoning. This was not needed for the core functionality of the game because it would increase complexity and the ability to leak information.

Timed levels or challenge modes were not added due to time constraints. The ability to challenge students is appreciated and keeps students engaged, however it was not added because of time constraints.

Finally, the ability to save progression was not added because it would increase complexity of the project. Each level does not require the previous to be completed because the next level would just require knowledge of the previously learned gates.

3 Specific Requirements

1. The software must be a game
    1.1. The software must take about 30 minutes to complete
    1.2. The software must have clear instructions
    1.3. The software must be able to be playable with keyboard and mouse
        1.3.1. The movement must use industry standard controls for gaming
    1.4. The software must include logic gates and wires
        1.4.1. The software should have accurate logic gates
        1.4.2. Logic gates should include NOT, OR, AND, NOR, NAND, XOR, and XNOR gates.
        1.4.3. The software must accurately demonstrate how each logic gate works
        1.4.4. The software must indicate when gates are recursively wired
        1.4.5. The wires will be placed to connect gates together
        1.4.6. The gates upon input must update their output
        1.4.7. The game must allow the player to place logic circuits
        1.4.8. The game must allow the player to place wires
    1.5. The software should be playable at 30 fps on a 4 core, 8 GB ram, and GTX 1050
    1.6. The software should be runnable on Windows 10 and Linux
    1.7. The software must have a main menu
        1.7.1. The main menu must have a play game, level select, credits, reference menu, and exit game options
        1.7.2. The reference menu must show how each logic gate works
        1.7.3. The credits section should should everyone that has worked on the game
        1.7.4. The play game must begin the first level
    1.8. The software must have levels
        1.8.1. The software must have a level selection menu
            1.8.1.1. The level selection menu must show all levels
            1.8.1.2. The level selection menu will not have level progressions, so every level should be available
    1.9. The software must be a top down game
        1.9.1. The game must allow the player to navigate around the map
    1.10. The software must give feedback upon successful completion of a level

1.10.1. The software must show a screen to proceed to the next level, select a new level or go to the main menu

1.11. The software must include a way to build logic circuits

2. The software must be educational for middle school students

2.1. The software must adhere to the Massachusetts DESE Curriculum DLSC Framework

2.1.1. Computing Systems [CS] by using a variety of computing devices to manipulate data.

2.1.2. Computational Thinking [CT] by using Abstraction and Algorithms

2.1.2.1. Design solutions that use repetition and conditionals

2.1.2.2. Define a simple function that represents a more complex task/problem and can be reused to solve similar tasks/problems

2.2. The software must use a learning progression

2.2.1. The software must use tutorials to teach the students how to use the game

2.2.2. The software must teach about different circuits with progression

2.3. The software must satisfy the requirements of an ESRB rating of E 10+.

2.4. The software must educate the user on how to build logic circuits

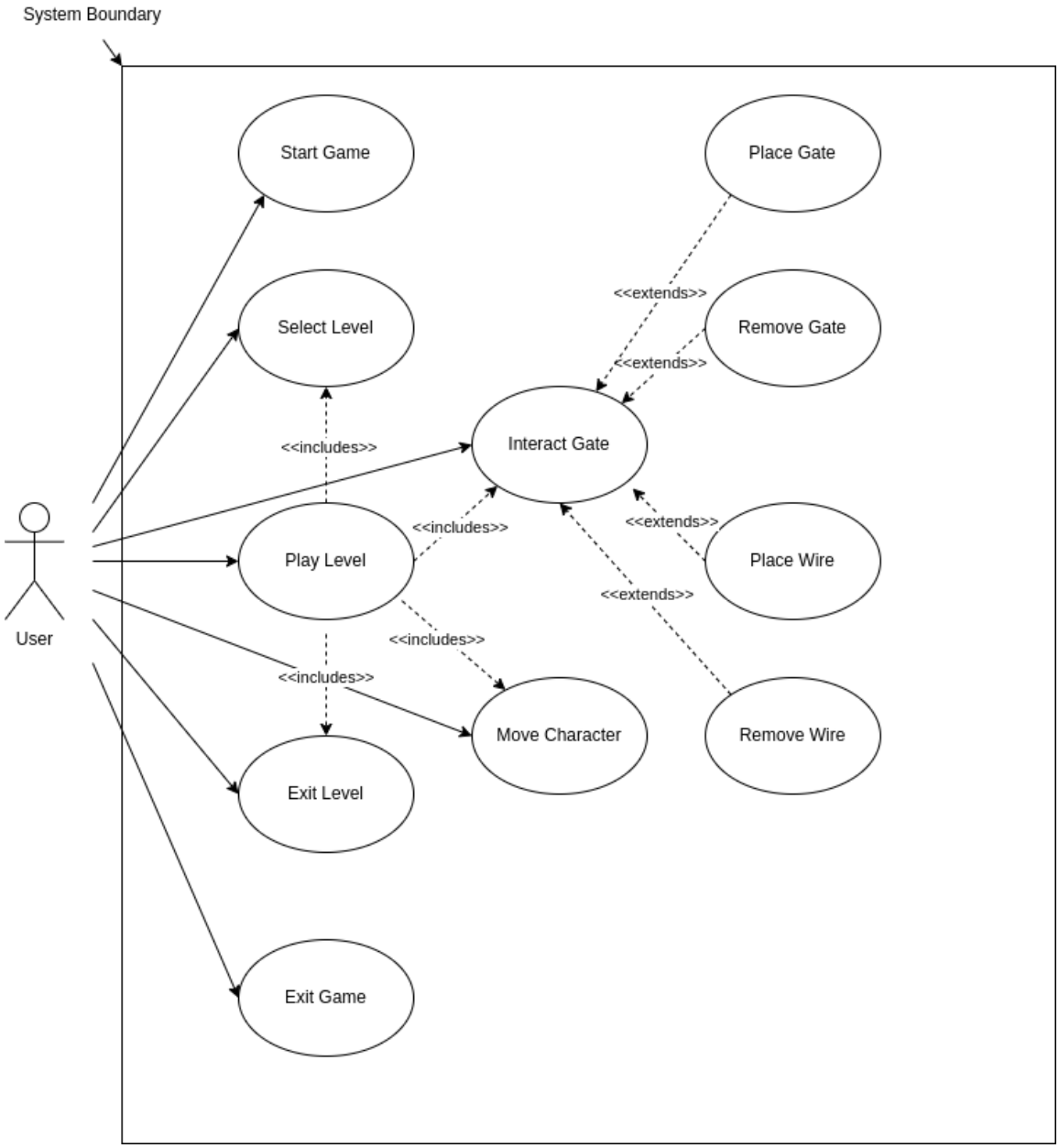
4 Modeling Requirements

4.1 Use Case Diagrams

Use Case Diagrams describe the behavior of the game, and how the user interacts with it. Each use case is specified, as well as their interactions. Actors are those who interact with the game.

The UML Use Case Diagram format represents use cases using ovals. Actors are represented using stick figures. Solid arrows pointing from an actor to a use case indicate use cases that the user directly interacts with; those without such an arrow are implementation details. Use cases may include other use cases, indicated with a dashed arrow labeled «includes». Use cases that are extensions of other use cases are indicated by a dashed line labeled «extends».

Figure 4.1.1 – Use Case Diagram



Use Case Name:	**Start Game**
Actors:	User
Description:	The user clicks on play game from the main menu to navigate to the first level.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1
Uses cases:	Used when the player wants to start the game without picking a level.

Use Case Name:	**Level Select**
Actors:	User
Description:	The user must click on Level Select on the main menu or after completion of a level to bring up a screen to select a level from a list.
Type:	Primary
Includes:	Play Level
Extends:	N/A
Cross-refs:	Requirement 1.8, Requirement 1.8.1
Uses cases:	Used when the player would like to select a level of their choosing.

Use Case Name:	**Play Level**
Actors:	User
Description:	The user either presses start game or select level to the play level.
Type:	Primary
Includes:	Level Select, Exit Level, Move Character, Interact Gates
Extends:	N/A
Cross-refs:	Requirement 1.8
Uses cases:	Used whenever the user would like to begin a level.

Use Case Name:	**Exit Level**
Actors:	User
Description:	The user presses the pause button and clicks exit or after completion of a level this will be activated
Type:	Primary
Includes:	Play Level
Extends:	N/A
Cross-refs:	Requirement 1, Requirement 18.1
Uses cases:	Used when the player wants to exit a level they are currently playing or when they complete a level the user will automatically leave the level.

Use Case Name:	**Exit Game**
Actors:	User
Description:	The user presses the exit game button from the main menu or if the application is forcibly closed.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1
Uses cases:	Used when the player wants to exit the game.

Use Case Name:	**Interact Gate**
Actors:	User
Description:	The user can add/remove gates and wires.
Type:	Primary
Includes:	Play Level
Extends:	Place Gate, Remove Gate, Place Wire, Remove Wire
Cross-refs:	Requirement 1.4.7
Uses cases:	Used when the player wants to pick up and place gates and wires to try and solve the puzzle.

Use Case Name:	**Move Character**
Actors:	User
Description:	The user presses any of the navigation keys.
Type:	Primary
Includes:	Play Level
Extends:	N/A
Cross-refs:	Requirements 1.9.1.
Uses cases:	Used when the player wants to navigate the scene.

Use Case Name:	**Remove Gate**
Actors:	User
Description:	The player must navigate over the gate they want to pick up and must press the pickup key.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.4.7 Requirement 1.4
Uses cases:	Used when the player wants to remove a gate from the scene.

Use Case Name:	**Place Gate**
Actors:	N/A
Description:	The player must navigate to the position they want to place a gate, then press the place gate button.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.4.7, Requirement 1.4
Uses cases:	Used when the player wants to move a gate.

Use Case Name:	**Place Wire**
Actors:	N/A

Use Case Name:	Place Wire
Description:	The player must navigate to where they want to place a wire and look at the appropriate input with the wire tool selected, then press the place wire button.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.4.8, Requirement 1.4
Uses cases:	Used when the player wants to connect gates to each other.

Use Case Name:	Remove Wire
Actors:	N/A
Description:	The player must navigate to where they want to place a wire and look at the appropriate wire with the wire tool selected, then press the remove wire button.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.4.8, Requirement 1.4
Uses cases:	Used when the player wants to either remove a connection between gates, or remove a wire on the ground.

4.2 Class Diagrams

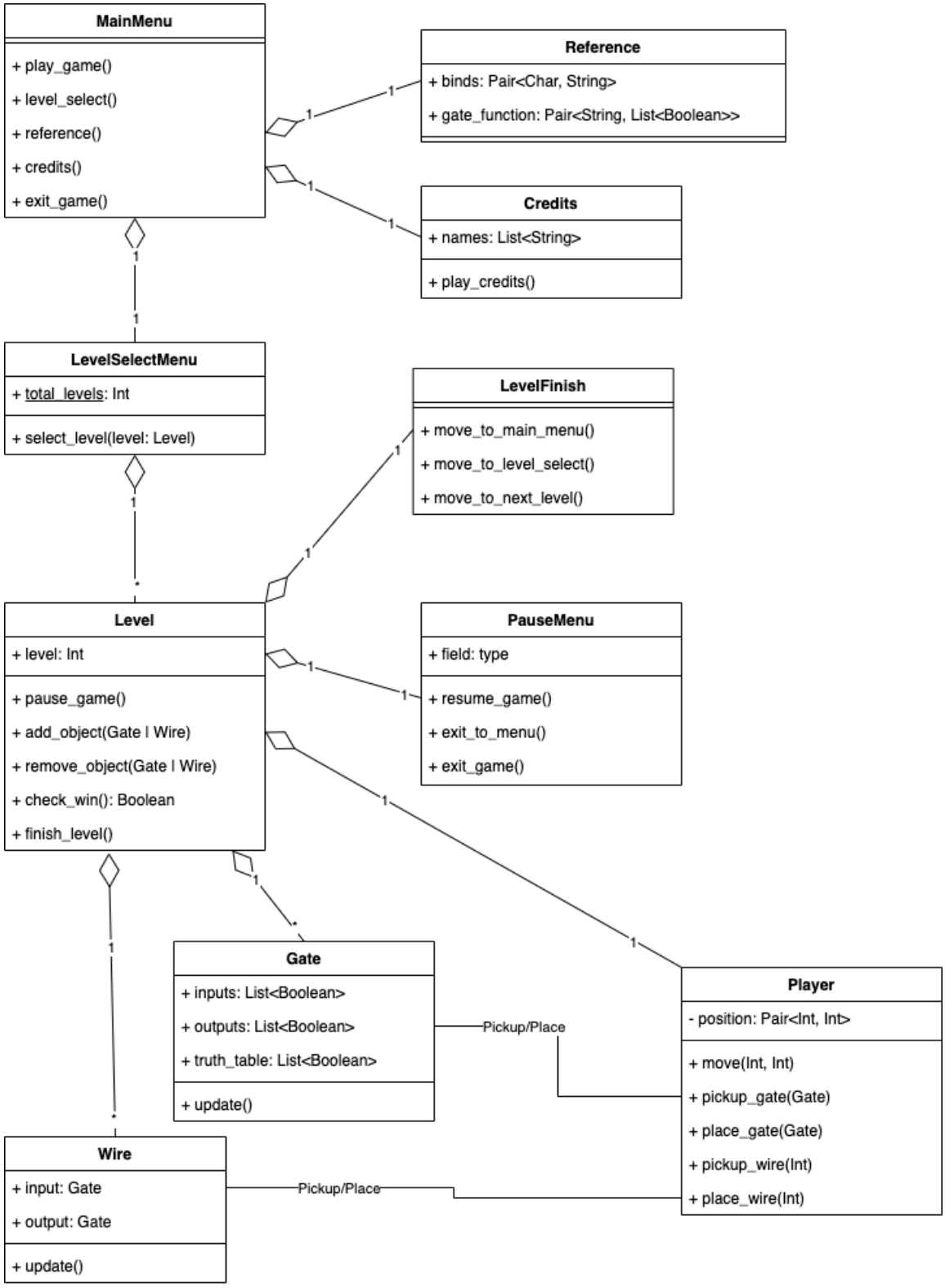
Class diagrams describe the structure of the game, and what parts are associated with one another. Classes describe what data objects hold, and what actions they can take.

UML Class diagrams represent classes as boxes with three major sections: the class name, the class members, and the class methods. The class name is put in **bold**, with interfaces represented by the tag «interface».

The class members list contains one entry per member, with each member containing a visibility specifier, a member name, and a type. The visibility specifier is either ‘+’ (public) or ‘-’ (private). The member name is underlined for static members, and not underlined for non-static members. The type is placed after the member name, separated by a colon.

The method list contains one entry per method, with each method containing a visibility specifier (as described above), a method name, a parameter list, and a return type. method names are underlined for static methods, and not underlined for non-static members. Parameters are a comma-separated list names and types, placed in parentheses. Each parameter contains a name and type, separated by a colon. The return type is specified after the parameters list, following a colon. The return type is omitted if the method returns nothing.

Figure 4.2.1 – Class Diagram



Class: MainMenu	
Class Description: This class is used to draw and process inputs for the Main Menu the game starts into.	
Class Members	**Member Description**
Class Methods	**Method Description**
+ play_game()	Transitions to the first level of the game.
+ level_select()	Transitions to the Level Select menu.
+ reference()	Transitions to the How to Play/Tutorial/Reference menu.
+ credits()	Transitions to the Credits menu.
+ exit_game()	Exits the game, terminating the program.

Class: Reference	
Class Description: This class is used to draw and process inputs for the How to Play / Gate Reference menu.	
Class Members	**Member Description**
+ binds: List<Pair<Char, String>>	Stores the keybinds of the game, to be presented to the user.
+ gate_function: List<Pair<String, List<Boolean>>>	Stores the gate mapping functions, so the gates may accurately be displayed to the user.
Class Methods	**Method Description**

Class: Credits	
Class Description: This class is used to draw and process inputs for the Credits menu.	
Class Members	**Member Description**
+ names: List<String>	Stores the names of people who helped with the game, so they can be listed in the credits.
Class Methods	**Method Description**
+ play_credits()	Plays the credits.

Class: LevelSelectMenu	
Class Description: This class is used to draw and process inputs for the Level Select menu, which allows the player to select a level to play.	
Class Members	**Member Description**
+ total_levels: Int	The number of total levels that exist in the game.
Class Methods	**Method Description**
+ select_level(level: Level)	Transitions to the selected level, specified by the parameter.

Class: Level
Class Description: This class is used to draw and process inputs for a Level, including adding/removing objects, pausing and resuming the game, processing logic

Class: Level	
signals, and detecting victory.	
Class Members	**Member Description**
+ level: Int	Stores the level ID of the level.
Class Methods	**Method Description**
+ pause_game()	Pauses the game, preventing player actions and opening a Pause menu.
+ add_object(obj: Gate | Wire)	Adds an object to the level.
+ remove_object(obj: Gate | Wire()	Removes an object from the level.
+ check_win(): Boolean	Checks to see if the level's victory condition is met.
+ finish_level()	Opens the Level Finish menu, providing options once the victory condition has been met.

Class: LevelFinish	
Class Description: This class is used to draw and process inputs for the Level Finish menu, which provides the user options once they have completed a level.	
Class Members	**Member Description**
Class Methods	**Method Description**
+ move_to_main_menu()	Navigates the user to the Main Menu.
+ move_to_level_select()	Navigates the user to the Level Select menu.
+ move_to_next_level()	Navigates the user to the next level.

Class: PauseMenu	
Class Description: This class is used to draw and process inputs for the Pause menu, which provides the user options during the game.	
Class Members	**Member Description**
Class Methods	**Method Description**
+ resume_game()	Returns to the game, closing the Pause menu.
+ exit_to_menu()	Navigates the user to the Main Menu, closing the current level.
+ exit_game()	Exits the game, returning the user to the desktop.

Class: Player	
Class Description: This class is used to draw and handle inputs for the Player, including movement, selecting gates and wires in the level, and placing and removing gates and wires.	
Class Members	**Member Description**
- position: Pair<Int, Int>	Stores the player's current position in the level.
Class Methods	**Method Description**
+ move(x: Int, y: Int)	Moves the player within a level. The player is moved *by* the specified coordinates, not *to* them.
+ pickup_gate(gate: Gate)	Removes a gate from the level.

Class: Player	
+ place_gate(gate: Gate)	Places a gate within the level. The gate is placed at the player's current position, aligned to the current tile grid.
+ pickup_wire(i: Int)	Removes a piece of wire from the level.
+ place_wire(i: Int)	Places a piece of wire within the level. The wire is placed at the player's current position, and connects to gates' inputs and outputs.

Class: Gate	
Class Description: This class is used to draw and update logic for gates placed within a level.	
Class Members	**Member Description**
+ inputs: List<Boolean>	Stores the current states of the inputs, where True indicates that the state is active, and False indicates that the state is inactive.
+ outputs: List<Boolean>	Stores the current states of the outputs, where True indicates that the state is active, and False indicates that the state is inactive.
+ truth_table: List<Boolean>	Stores the truth table for this gate. This table is indexed by the numerical representation of the inputs, and stores the output value as a Boolean.
Class Methods	**Method Description**
+ update()	Updates the outputs to reflect the current state of the inputs, using the truth table.

Class: Wire	
Class Description: This class is used to draw and update logic for wires placed within a level.	
Class Members	**Member Description**
+ input: Gate	Stores the input gate that this piece of wire receives a signal from.
+ output: Gate	Stores the output gate that this piece of wire sends a signal to.
Class Methods	**Method Description**
+ update()	Updates the input of the output gate to reflect the output of the input gate.

4.3 Sequence Diagrams

Sequence diagrams describe the order of events within the game. Each diagram lists the method calls required to carry out a specific task.

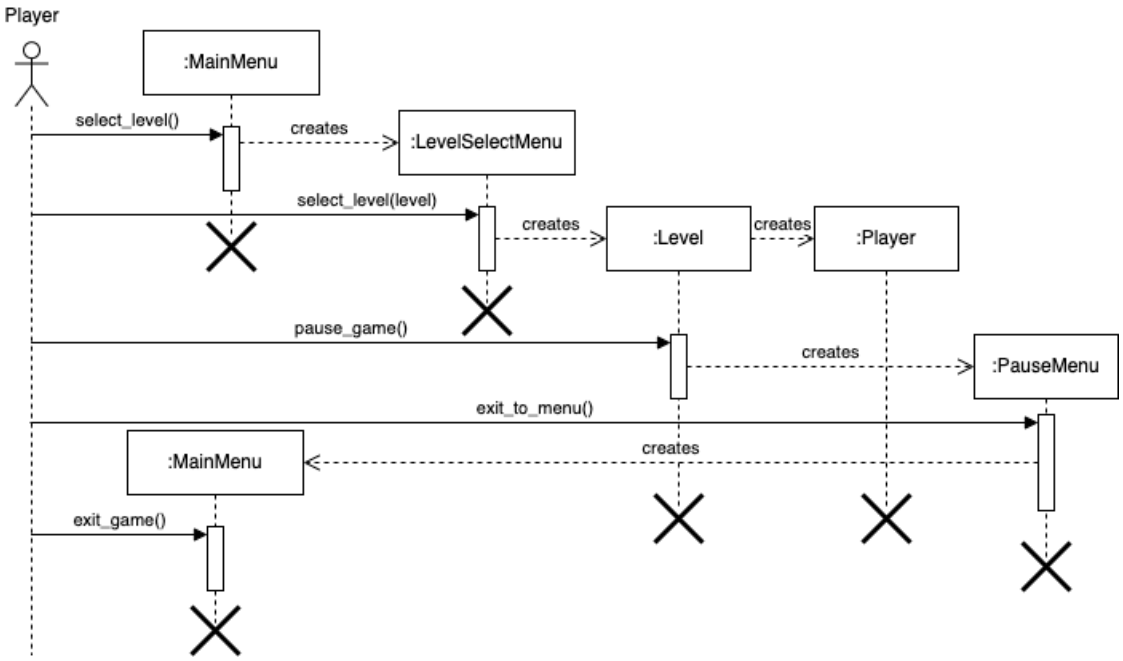
UML sequence diagrams represent actors and classes with a vertical line, called a lifeline. Lifelines for classes contain a box with the name of the class at the top, and actor lifetimes contain a stick figure at the top. Lifelines are terminated with a ‘X’.

Solid arrows from one lifeline to another represent method calls. Calls create a vertical bar along the receiver’s lifetime, from which subroutines can be invoked. Dashed lines labeled ‘creates’ point to class boxes, creating a new lifeline. Dashed lines labeled ‘return’ indicate returns from method calls, and may be omitted if nothing is returned.

Frames labeled ‘loop’ will be executed multiple times, until the loop condition is met. The loop condition is placed in brackets below the frame label.

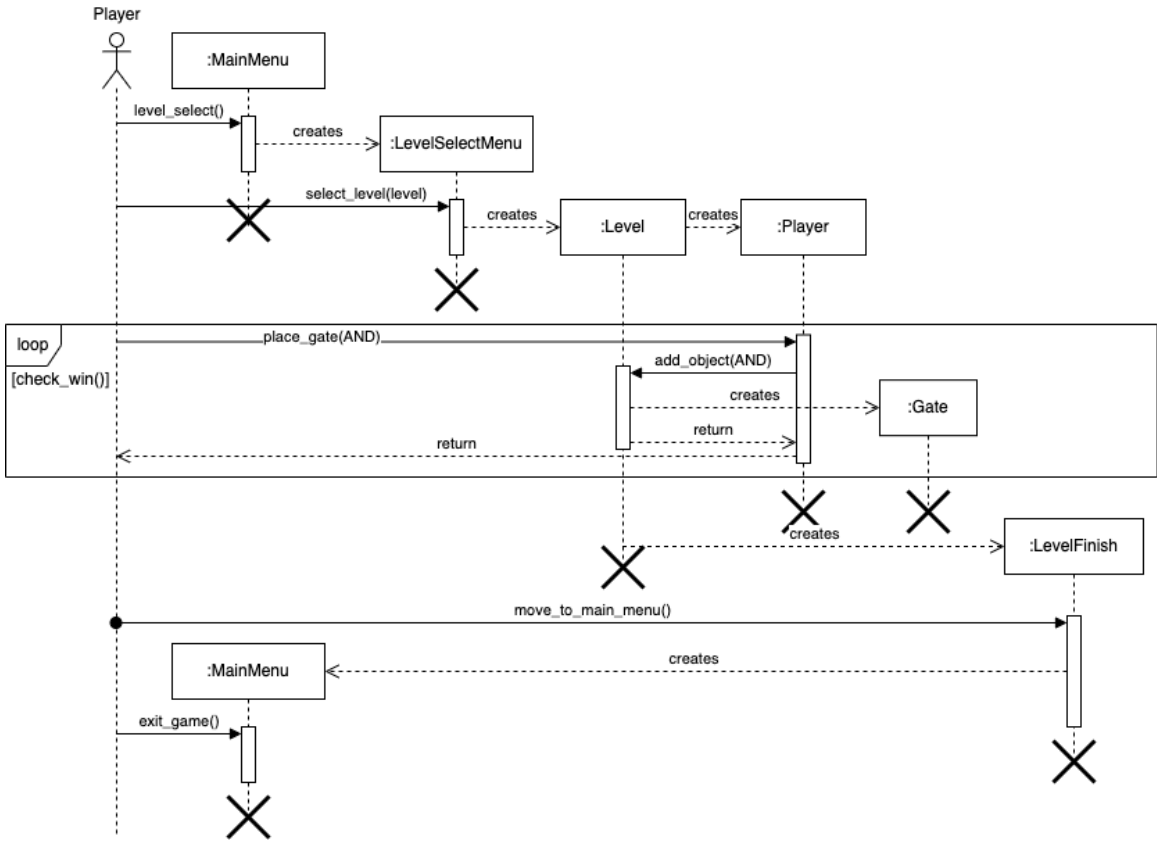
In this sequence, the player has opened the game and is met with the main menu. The player presses the “Level Select” button to navigate to the Level Select menu. The player then selects a level to play. Once the level starts, the player presses the “Escape” key to pause the game, and presses the “Main Menu” button to return to the Main Menu. The player then presses the “Exit” button to quit to the desktop.

Figure 4.3.1 – Sequence Diagram 1



In this sequence, the player has opened the game and is met with the main menu. The player presses the “Level Select” button to navigate to the Level Select menu. The player then selects a level to play. Once the level starts, the player places an AND gate. This completes the level, so the Level Solved menu opens. The player selects “Back to Main Menu”, returning to the Main Menu. The player then presses the “Exit” button to quit to the desktop.

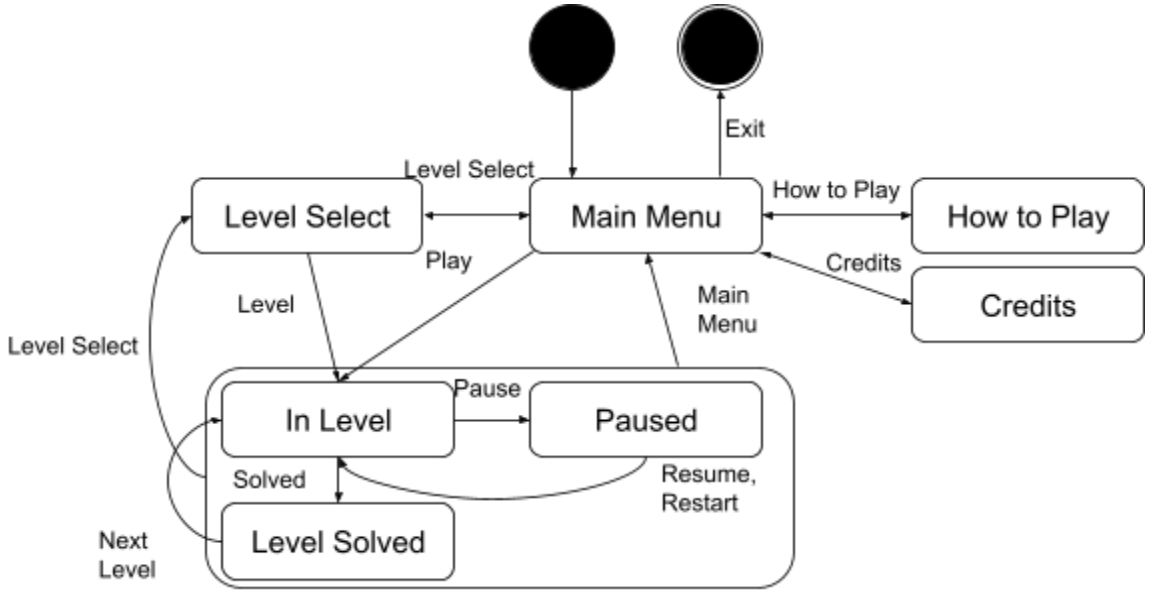
Figure 4.3.2 – Sequence Diagram 2



4.4 State Diagrams

State diagrams describe the navigation flow of the game. States encode the current location within the game, and when to move between states.

State diagrams represent states with labeled rounded rectangles. States have labeled arrows, indicating when the game transitions to a different state. The “In Level”, “Paused”, and “Level Solved” states are grouped into one larger state. The solid circle represents the start state, which will point to the initial state of the game. The solid circle with an outline represents the end state, which, when navigated to, terminates the program.



5 Prototype

The prototype will demonstrate the navigation and controls for the game, including the following:

- Main Menu
- Level Select Menu
- Pause Menu
- Level Solved Menu
- How to Play Menu
- Credits Menu
- Moving the Player
- Toggling Switches
- Completing Levels

5.1 How to Run Prototype

5.1.1 Windows

- Navigate to the GitHub Releases page: https://github.com/NateWright/LeveledLogic/releases
- Locate the latest release (v0.1.0), and download the file “Windows.Desktop.zip”.
- Unzip the zip file, and run the executable file contained within.
- When Windows SmartScreen blocks the app launch, click “More Info” and then “Run Anyways”.

5.1.2 Linux

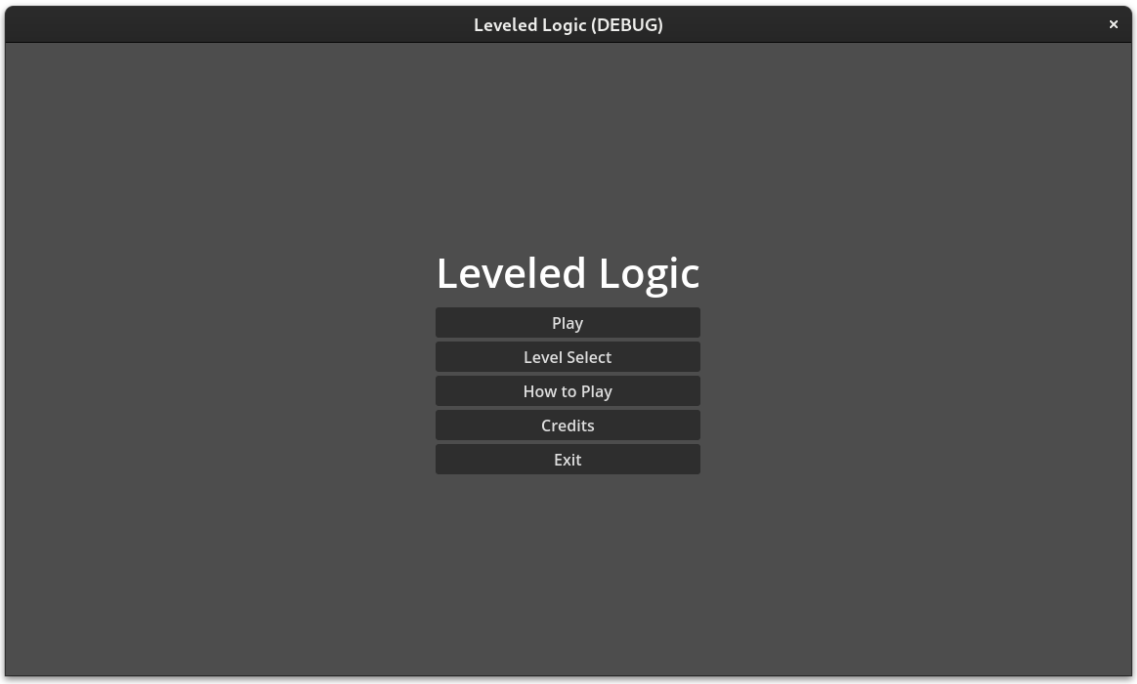
- Install Flatpak for your distribution: https://flathub.org/setup
- Run the following commands in your terminal:

```
wget https://raw.githubusercontent.com/NateWright/LeveledLogic/main/com.github.NateWright.LeveledLogic.pgp
flatpak remote-add --gpg-import=com.github.NateWright.LeveledLogic.pgp LeveledLogic https://natewright.github.io/LeveledLogic/
flatpak install LeveledLogic com.github.NateWright.LeveledLogic
flatpak run com.github.NateWright.LeveledLogic
```

5.2 Sample Scenarios

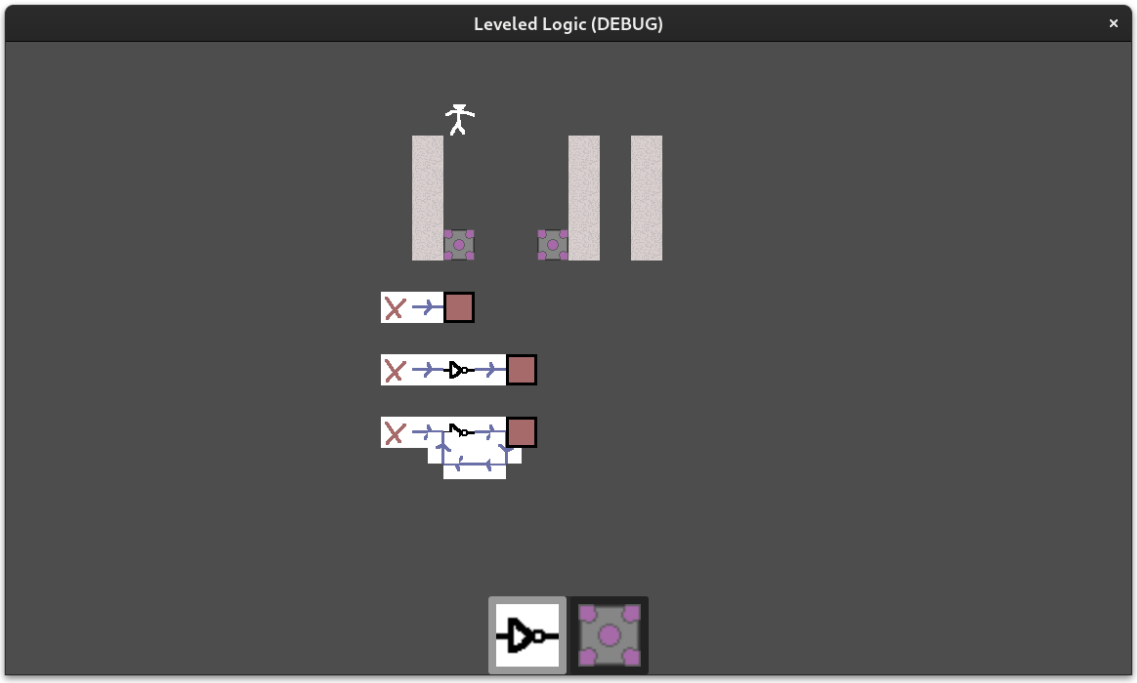
The Player opens the app and is met with the Main Menu:

Figure 5.2.1 – Main Menu



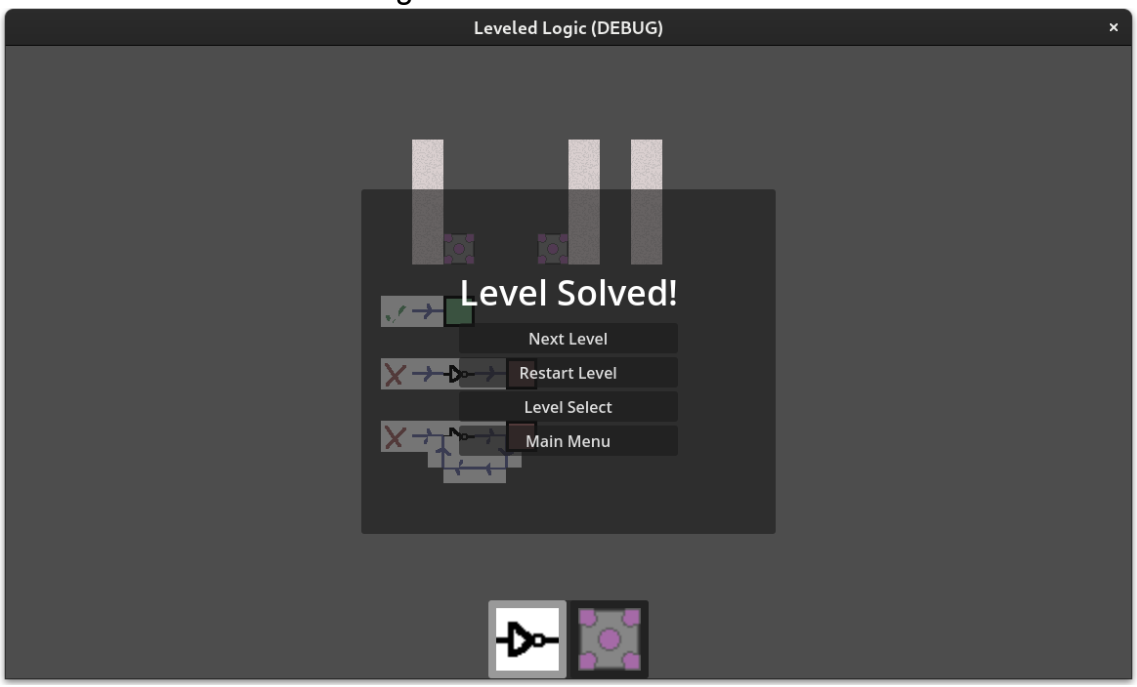
The Player selects “Play”, bringing them to the first (test) level:

Figure 5.2.2 – Test Level



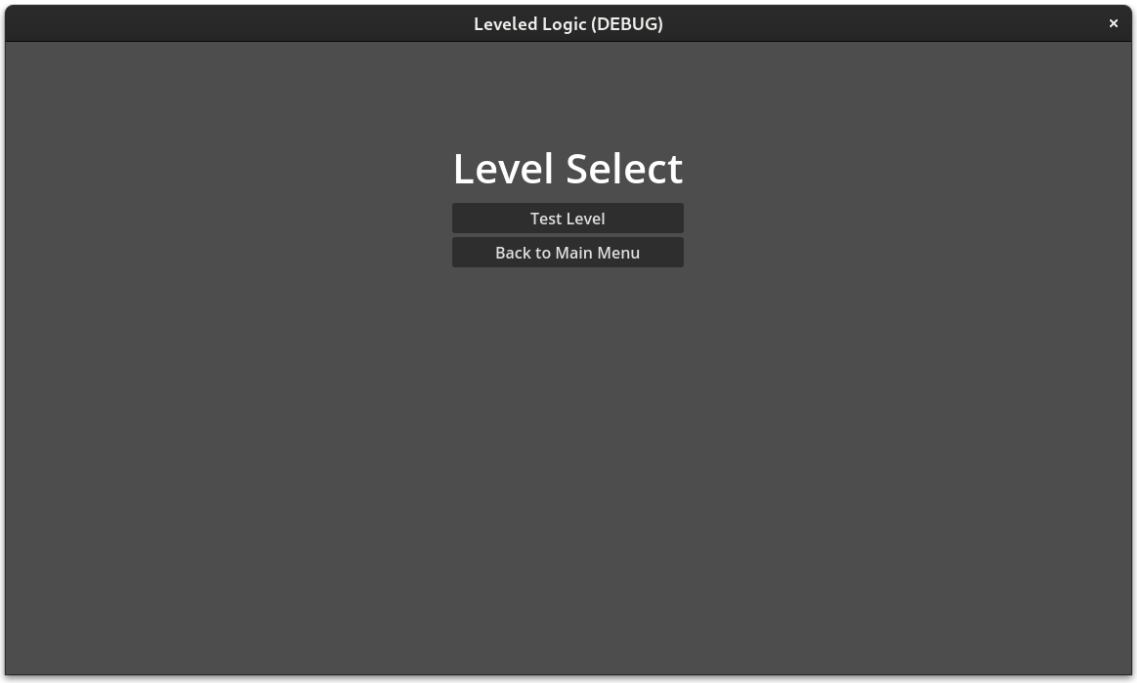
The Player moves to the top switch and presses the “Space” key to toggle it, solving the level:

Figure 5.2.3 – Level Solved



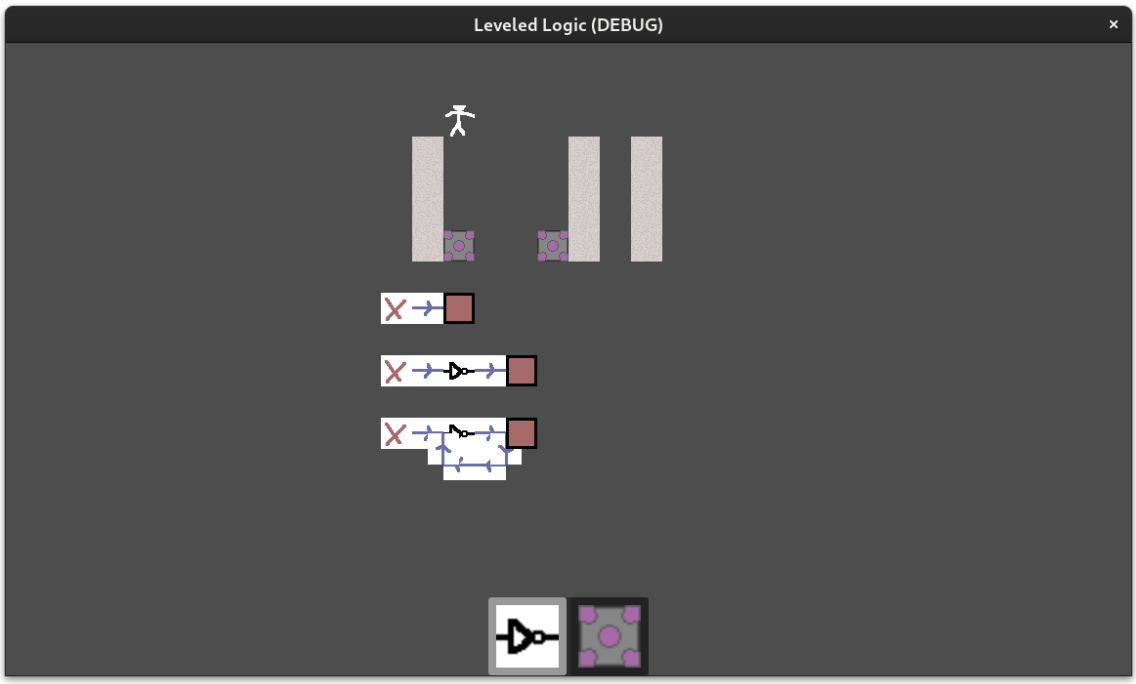
The Player selects “Level Select”, bringing them to the level select menu:

Figure 5.2.4 – Level Select



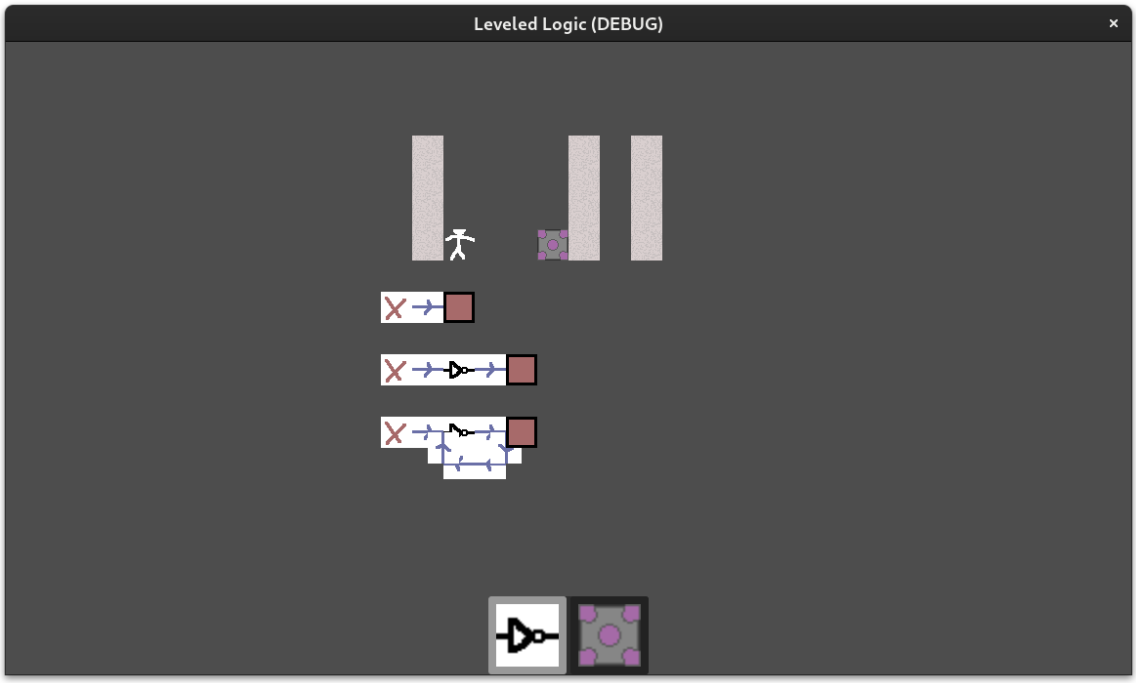
The Player selects “Test Level”, bringing them to the test level again:

Figure 5.2.5 – Test Level



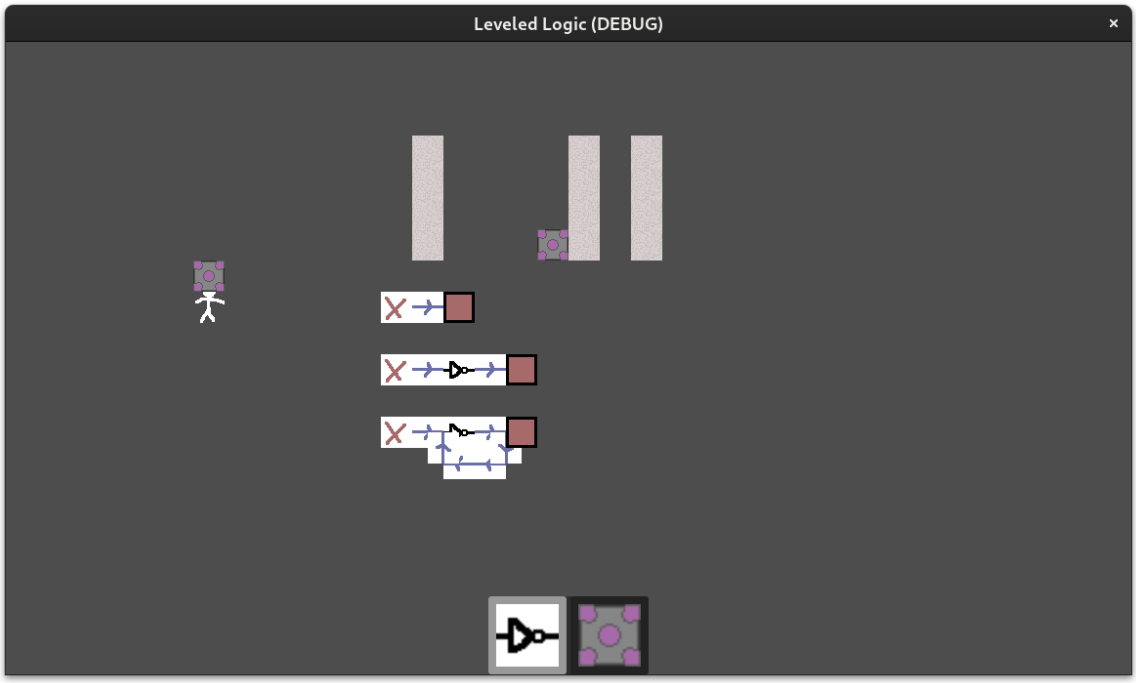
The Player walks to the picks up an object, and presses “Shift+E” to pick it up:

Figure 5.2.6 – Picking up an object



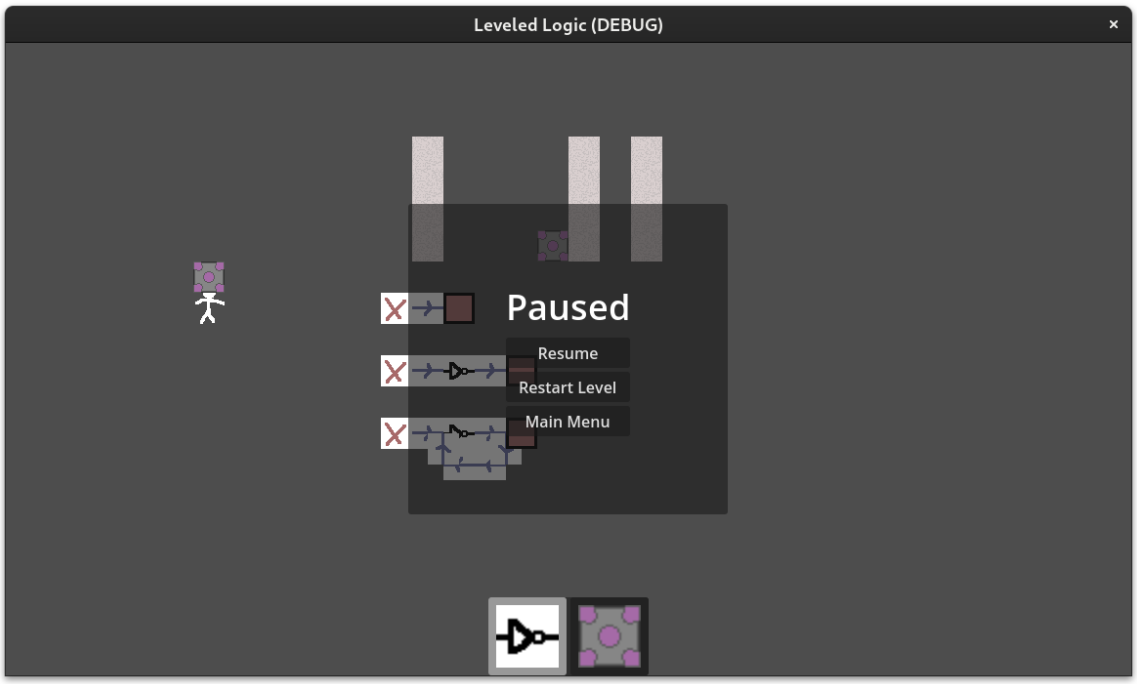
The Player moves to another location, and presses “E” to place the object elsewhere:

Figure 5.2.7 – Placing an Object



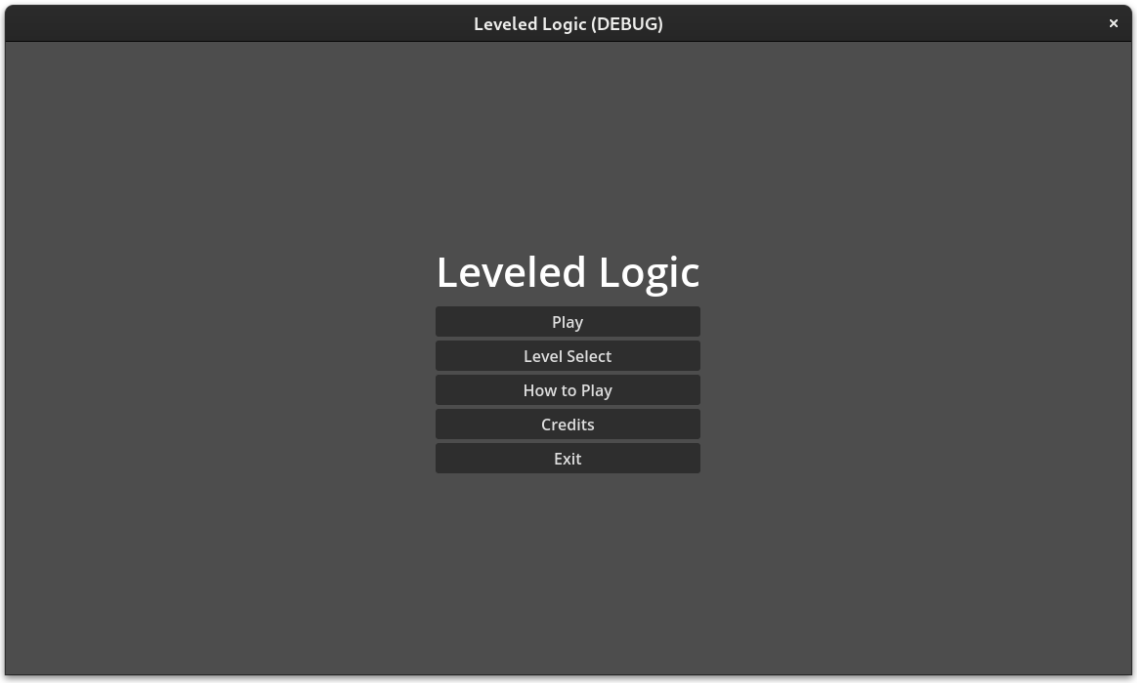
The Player presses the “Escape” key to pause the game:

Figure 5.2.8 – Pause Menu



The Player presses the “Main Menu” button to return to the Main Menu

Figure 5.2.9 – Main Menu



Player presses the “Exit” button to exit the game.

6 References

[1] D. Thakore and S. Biswas, “Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks,” Proceedings of IEEE Military Communication, Atlantic City, October 2005.

[2] J. Linietsky, A. Manzur, et al., “Godot Engine 4.1 Documentation” Godot Engine. https://docs.godotengine.org/en/stable/index.html (accessed November 17, 2023).

[3] S. Bourassa, S. Calla, A. Cheang, J. Rathore, and A. Rosa, “Software Requirements Specification (SRS) Project Blackboard Bookmarking Bar,” Team: 2, 2023.

[4] Massachusetts Department of Elementary and Secondary Education. (2016, October). *Digital Literacy and Computer Science Massachusetts Curriculum Framework – 2016.* [Online]. Available: https://www.doe.mass.edu/frameworks/dlcs.pdf

[5] N. Wright, J. Kang, G. Shahrouzi, E. Ta, C. Klein, “Leveled Logic” Team 8. https://natewright.github.io/LeveledLogicWebsite/ (accessed November 17, 2023).

[6] B. Ikenaga, “Truth Tables,” Millersville University, 2023. [Online]. Available: https://sites.millersville.edu/bikenaga/math-proof/truth-tables/truth-tables.html.[Accessed: Nov. 20, 2023].

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.