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

LeveledLogic is a logic gate simulation where the idea is to connect wires from inputs to gates and then finally to the outputs. The inputs and outputs are determined via truth tables and different levels accompany different requirements as to what the user should do.

In section 1, we provide an introduction which identifies the purpose and scope of Leveled Logic and provides definitions, acronyms, abbreviations, and organization for Leveled Logic.

In section 2, we give an overall description which gives a brief introduction of what information is covered in this section.

In section 3, we give specific requirements which gives an enumerated list of requirements which uses a hierarchical numbering scheme for Leveled Logic.

In section 4, we provide modeling requirements which specifies the bridge between the application domain and the machine domain and describes the use and notation for each use case diagram.

In section 5, we give a prototype which describes what the prototype shows in terms of system functionality and how to run the prototype.

In section 6, we provide a list of all documents referenced in the SRS by identifying each document by title, report number, date, and publishing organization and specifying the sources from which the references can be obtained (includes the project website).

In section 7, we give a point of contact which 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 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 software project is Leveled Logic which is made using Godot. The application domain is educational and designed for elementary and middle schoolers. Leveled Logic is a logic simulator game. The game aims to improve problem solving abilities and abstract reasoning.

One goal includes making it easy to use for teachers. By easy to easy, we mean making kid friendly and allowing teachers to run the software on the majority of their machines. Kid friendly implies adhering to certain constraints we placed on the software. As a result, teachers do not have to fret that the material is not appropriate for their students nor do they have to jump through hoops to run the software.

Elementary and middle schoolers have the ability to freely interact with the environment containing logic gates and wires. This is a rare occurrence for most as they do not study the theory behind circuits until high school and sometimes even university. As a result, they have the ability to experiment with tools they never have come into contact with before.

1.3 Definitions, acronyms, and abbreviations

Term	Definition
True	A 1 in binary denoting an object is on.
False	A 0 in binary denoting an object is off.
Compound Statement	Composed by one or more simple statements [6].
Simple statement	Represented by symbols (often letters) [6].
Symbol	Represents a statement [6].
Statement	A declarative sentence which is either true or false.
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 [5].
Logic Gate (Gate)	They perform basic logical functions outlined by their truth tables.
NOT Gate	A gate with one input, where the output is the opposite of the input.
OR Gate	A gate with two inputs, where the output is true iff either of the two inputs is true, or if both are true.

Term	Definition
AND Gate	A gate with two inputs, where the output is true iff both inputs are true.
XOR Gate	A gate with two inputs, where the output is true iff either of the two inputs is true, but not if both are true.
NOR Gate	A gate with two inputs, where the output is false iff either of the two inputs is true, or if both are true. This is equivalent to an OR gate followed by a NOT gate.
NAND Gate	A gate with two inputs, where the output is false iff both inputs are true. This is equivalent to an AND gate followed by a NOT gate.
XNOR Gate	A gate with two inputs, where the output is false iff either of the two inputs is true, but not if both are true. This is equivalent to an XOR gate followed by a NOT gate.
Source	An input to a level. Players can connect sources to the inputs of gates.
Sink	An output to a level. Players can connect sinks to the outputs of gates.
Lamp	A block lights up when the input received is true and stays dark when the input received is false.
Gate input	The value that a gate receives, either from a source or from another gate.
Gate output	The value that a gate sends to either a sink or to another gate.
Filled gate	Each input for a gate is connected from either another gate or a source.
Circuit	A group that may consist of sources, sinks, logic gates, and wires.
Grid	The play area for each level is made up of cells.
Cell	A uniform square that the grid is made up of.
Wire	Connects two different logic gates together, carrying the output signal from one to the input signal of the other.

Term	Definition
Recursively Wired	A gate is recursively wired if one of its inputs is dependent on its output, either directly or through a chain of logic gates.
Level	An instance of the game that involves solving a task of putting together wires and gates.
Top-Down Perspective	Top-down perspective, also sometimes referred to as bird's-eye view, overhead view or helicopter view, when used in video games refers to a camera angle that shows the player and the area around them from above [8].
Player	The student playing the game.
Avatar	An instantiation representing the player that allows them to interact in the level via placing or removing gates and wires.
Hotbar	A toolbar at the bottom of the screen that allows the player to access gates and wires. Utilizing this, they are able to place or remove gates and wires.
Godot	The game engine used to design Leveled Logic.
GDScript	Godot’s built in scripting language.
Industry Standard Gaming Controls	WASD + Shift: Sprint or run faster.

Term	Acronym
UI	User interface
SW	Software
HW	Hardware
DESE	The Massachusetts Department of Elementary and Secondary Education

Term	Acronym
DESE Curriculum DLCS Framework	The DESE Digital Learning and Computer Science 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

The product perspective gives information how Leveled Logic is useful in an educational setting. More specifically, it shows the various skills that students strengthen while playing the game. Product functions discuss the intricacies of the game while providing a goal diagram to pictorially describe the prose. User characteristics provide details about the expected audience for the game and the level of experience they need to be comfortable using the software. Constraints lists important constraints that pertain to education centered towards elementary and middle school students. Assumptions and dependencies discuss what is assumed such as the hardware that is used to run the game. Apportioning of requirements details what the project lacked and to a greater detail what is going to be accomplished in later versions of the game.

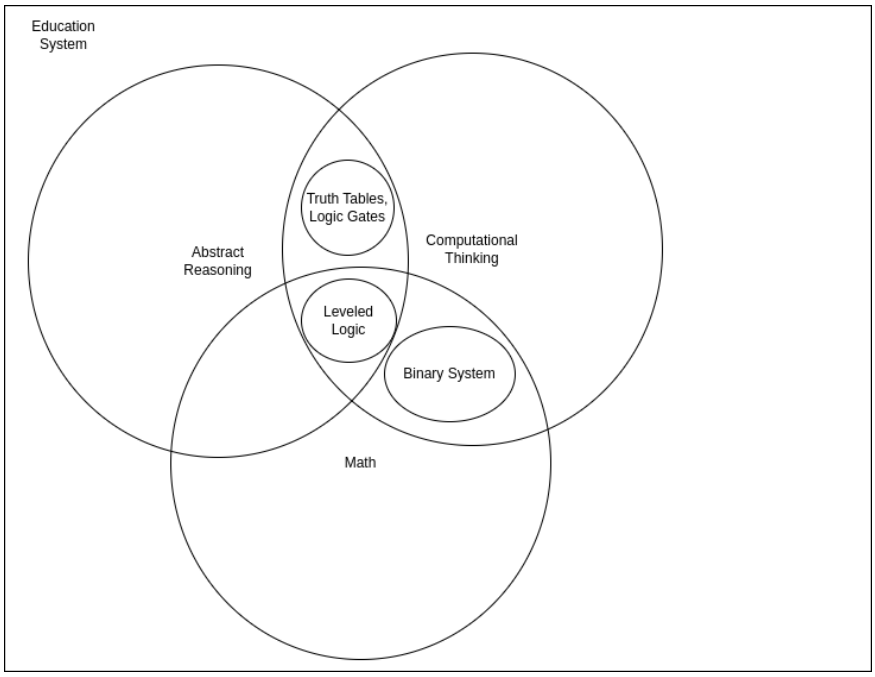
2.1 Product Perspective

Leveled Logic is a stand alone 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.

The context for the product is to be used in an educational setting designed to teach concepts within the field of Computer Science and more specifically logic theory. The game helps students improve their abstract reasoning, computational thinking, and math skills.

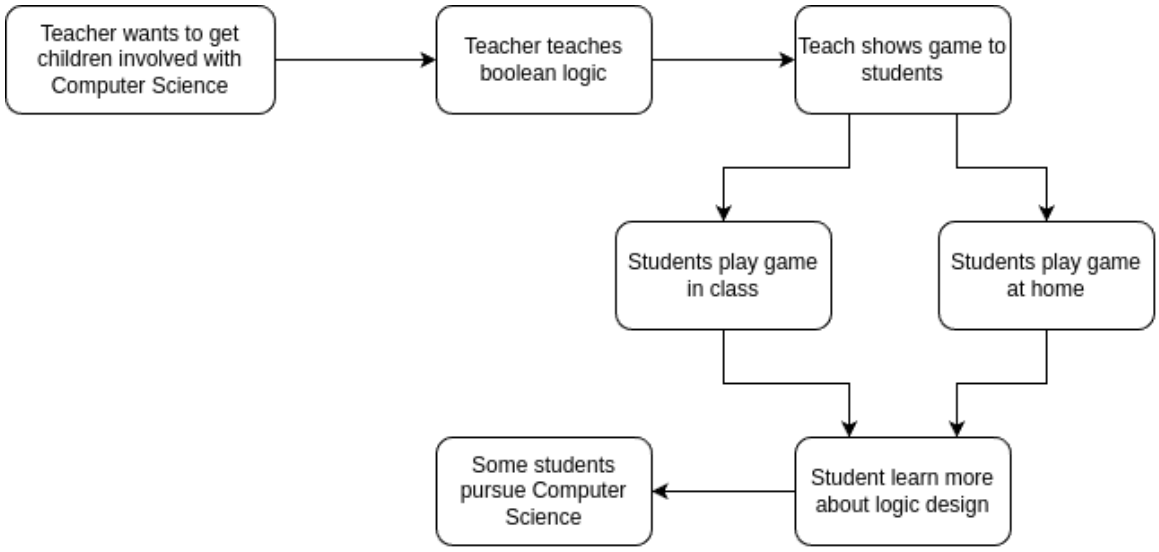
Figure 2.1.1 – Pictorial Diagram



2.2 Product Functions

Leveled Logic allows users to interact with the UI to select a level. Within the level, the player can place 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 elementary or middle school. 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.

Users are not expected to have formal knowledge in Computer Science as it is a topic that is not generally introduced until high school or at a post secondary institution. Specific topics such as the binary system may prove helpful but are not needed.

2.4 Constraints

An educational constraint is to follow the Massachusetts DESE Curriculum DLSC Framework. For instance, the game must utilize some form of computational thinking by using abstraction and algorithms [3]. According to this standard, students are allowed to interact with logic gates which helps develop an understanding of computer science and more specifically an understanding of circuit theory.

A constraint focused around making the game kid-friendly is to adhere to an ESRB rating of E 10+.

Leveled Logic does not have any security or safety-critical properties.

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 limited 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 is not 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 game must have levels
        1.1.1. The levels must have a top-down perspective
        1.1.2. The levels must use a grid as the background
            1.1.2.1. At most one object must occupy a cell in the grid
                1.1.2.1.1. The objects must be wires, logic gates, sources, or sinks
    1.2. The levels must include logic gates
        1.2.1. The levels must allow the avatar to place gates
        1.2.2. The levels must allow the avatar to remove gates
        1.2.3. The logic gates must include NOT, OR, AND, NOR, NAND, XOR, and XNOR gates.
        1.2.4. The gate's output must be updated if the gate is filled
    1.3. The levels must include wires
        1.3.1. The levels must allow the avatar to place wires
            1.3.1.1. The wires must be placed from an input such as a source or gate output to a sink or gate input
        1.3.2. The levels must allow the avatar to remove wires
        1.3.3. The levels must indicate when gates are recursively wired
    1.4. The levels must include a hotbar
        1.4.1. The hotbar must include logic gates
        1.4.2. The hotbar must include the option to add a wire
        1.4.3. The hotbar must include the option to remove a wire
    1.5. The levels must include an avatar
        1.5.1. The user must be able to move the avatar
    1.6. The levels must be completable
        1.6.1. The user must be able to check that their solution is correct
2. The software must have a UI
    2.1. The UI must have a main menu
        2.1.1. The main menu must have a play game, level select, credits, controls, truth tables, and exit game options
        2.1.2. The credits section must include everyone that has worked on the game

2.1.3. The play game must begin the first level

2.2. The UI must have a level selection menu

2.2.1. The level selection menu must show all levels

2.2.2. The level selection menu must have level progression

2.2.2.1. The player must start on the first level

2.2.2.2. The player must complete the prior level before starting the next level

2.3. The UI must have a controls menu

2.3.1. The menu must show the controls for certain actions

2.3.1.1. There must be a control to move the avatar

2.3.1.2. There must be a control to place or remove gates and wires

2.4. The UI must have a truth tables menu

2.4.1. The menu must show how each logic gate functions

2.4.1.1. Each logic gate must have an accompanying truth table

2.5. The UI must give feedback upon successful completion of a level

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

3. The software must be usable by the majority of people with computers

3.1. The software must be runnable on Windows 10 and Linux

3.2. The software must be playable at 30 FPS on a 4 core, 8 GB RAM, and GTX 1050

3.3. The software must be able to be playable with keyboard and mouse

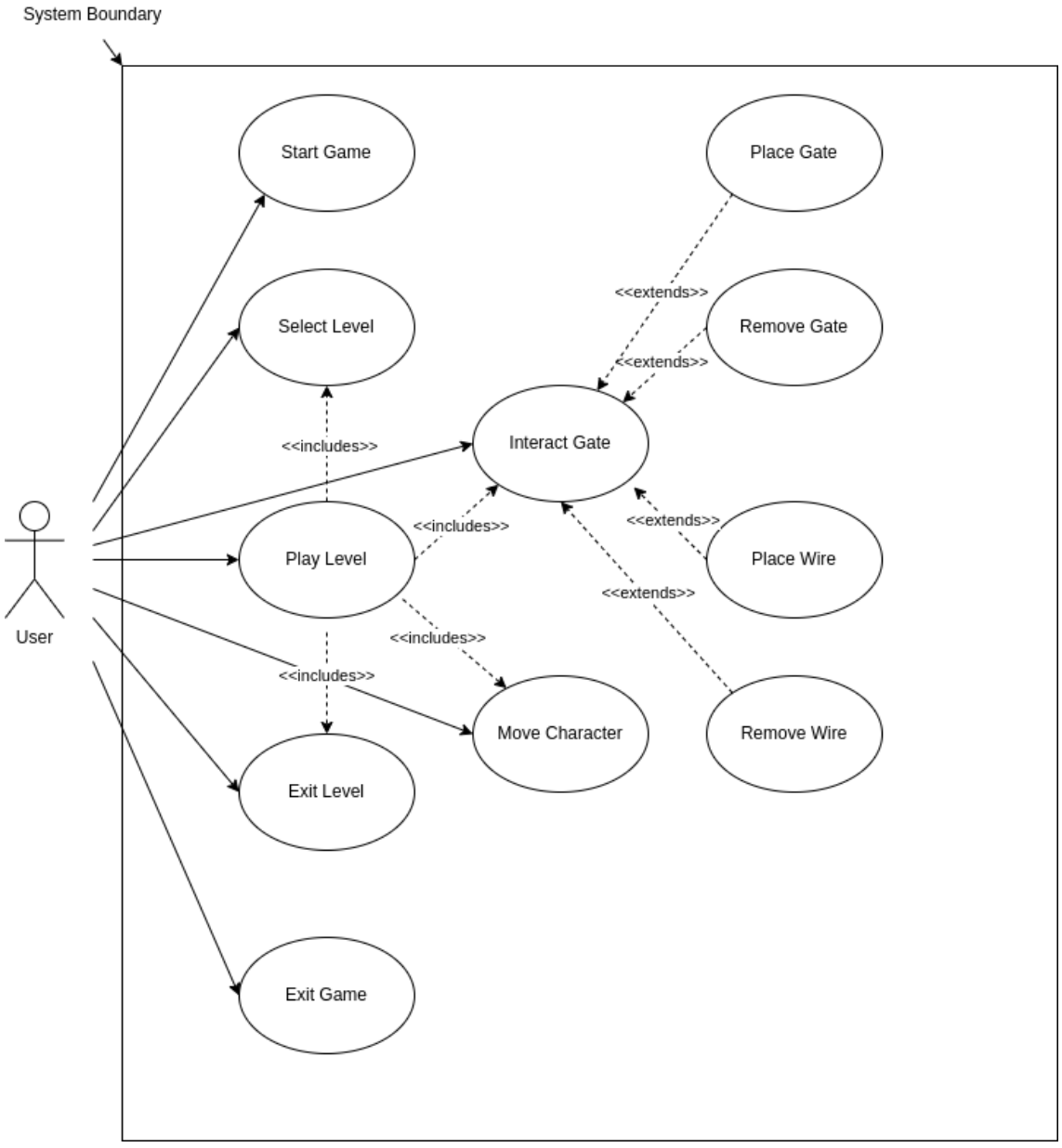
3.3.1. The movement must use industry standard controls for gaming

4 Modeling Requirements

4.1 Use Case Diagrams

The user can either play or exit the game where starting the game moves the user across a simple level progression starting from level 1 onwards to the last level. However, the user can also choose to select an arbitrary level. Once the user is in the game, they can then interact with gates and wires while moving their character across the grid. Additionally, the user can place and remove both gates and wires.

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 2.2
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
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.
Type:	Primary

Use Case Name:	**Exit Level**
Includes:	Play Level
Extends:	N/A
Cross-refs:	Requirement 2
Uses cases:	Used when the player wants to exit a level they are currently playing or when they complete a level the user automatically leaves 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 2.1.1
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 1.2.1, 1.2.2, 1.3.1, 1.3.2
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 1.5.1
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 remove and look at the appropriate gate with the remove gate tool selected and press the remove gate button.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.2.2
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 1.2.1
Uses cases:	Used when the player wants to place a gate.

Use Case Name:	**Place Wire**
Actors:	N/A
Description:	The player must navigate to where they want to place a wire and look at the appropriate input or output with the wire tool selected, then press the place wire button.
Type:	Secondary
Includes:	N/A
Extends:	Interact Gate
Cross-refs:	Requirement 1.3.1
Uses cases:	Used when the player wants to connect gates to each other.

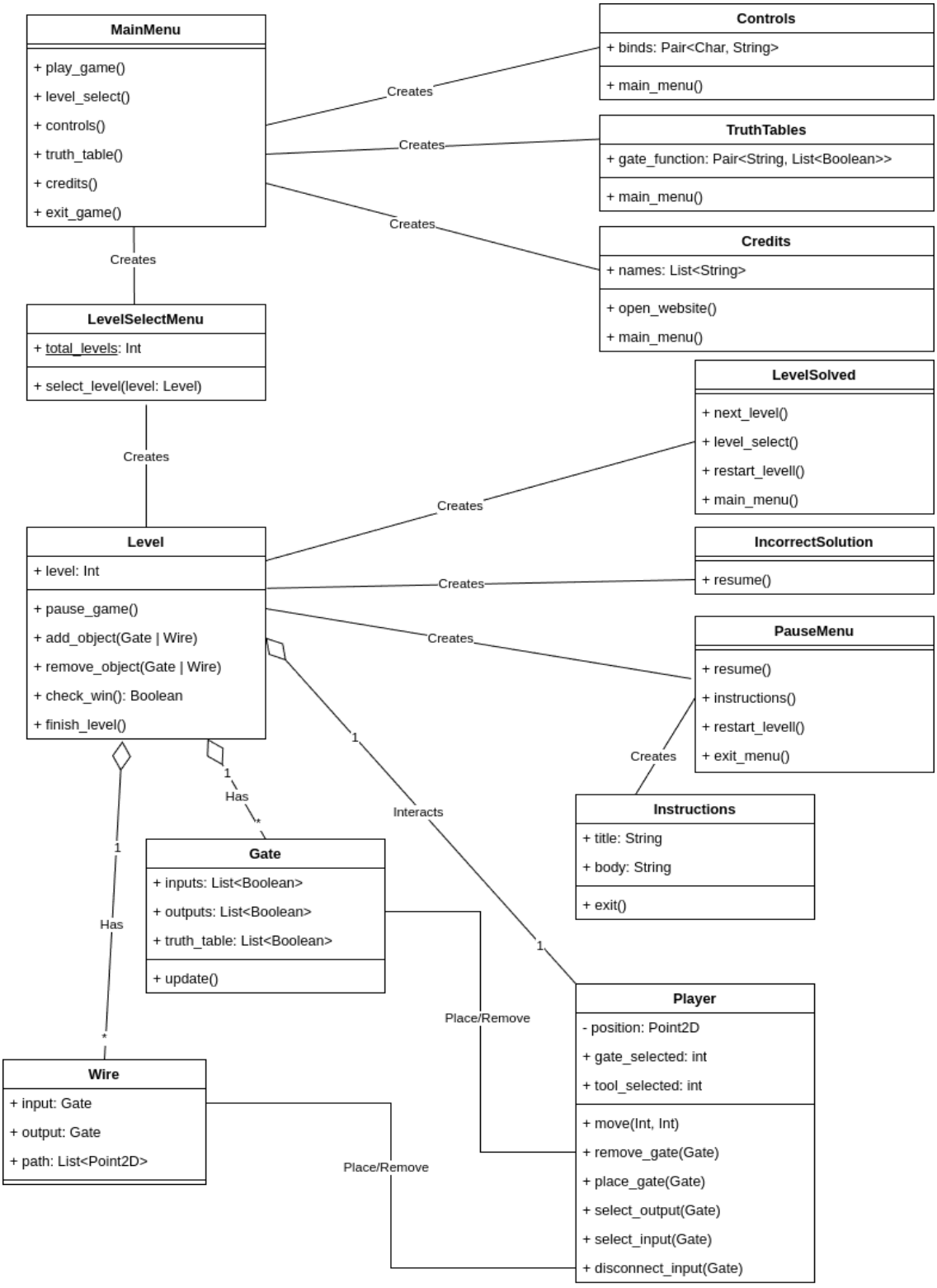
Use Case Name:	**Remove Wire**
Actors:	N/A
Description:	The player must navigate to a gate with a connected input and look at the appropriate input with the wire tool selected, then press the remove wire button.
Type:	Secondary
Includes:	N/A

Use Case Name:	**Remove Wire**
Extends:	Interact Gate
Cross-refs:	Requirement 1.3.2
Uses cases:	Used when the player wants to either remove a connection between gates.

4.2 Class Diagrams

The main menu uses various features such as a credits menu which allows the user to see information regarding that section. A level is made up of gates and wires where the player can modify them freely. Additionally, the level can be paused and when completed successfully, moves the user onto the next level. Users can interact with the level by pausing and testing out their current solution. Pausing gives the user the option to look at the instructions which can help them with their solution.

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 Methods	**Method Description**
+ play_game()	Transitions to the first level of the game.
+ level_select()	Transitions to the Level Select menu.
+ controls()	Transition to the Controls menu.
+ truth_table()	Transition to Truth Table menu.
+ credits()	Transitions to the Credits menu.
+ exit_game()	Exits the game, terminating the program.

Class: Controls (UI)	
Class Description: This class shows the controls for the game, including the controls to access avatar movement, the hotbar, the switch tool, and placing objects.	
Class Members	**Member Description**
+ binds: List<Pair<Char, String>>	Stores the keybinds of the game, to be presented to the user.
Class Methods	**Method Description**
+ main_menu()	Transitions the user to the Main Menu.

Truth Tables (UI)
Class Description: This class shows the respective truth table for each logic gate.

Class Members	Member Description
+ gate_function: List<Pair<String, List<Boolean>>>	The respective truth table for each logic gate.
+ main_menu()	Navigates the user to the Main Menu.

Class: Credits	
Class Description: This class shows the credits for the game including who worked on it and what their role was.	
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**
+ open_website()	Opens the Leveled Logic webpage in the web browser.
+ main_menu()	Navigates the user to the Main Menu.

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**

Class: LevelSelectMenu	
+ 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 signals, and detecting victory.	
Class Members	**Member Description**
+ level: Int	Stores the level ID of the level.
+ gates: List<Gate>	Stores a list of gates placed in the level.
+ wires: List<Wire>	Stores a list of wires connecting gates in 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. If it is met, the LevelSolved menu is shown, otherwise it displays IncorrectSolution.
+ finish_level()	Opens the LevelSolved menu, providing options once the victory condition has been met.

Class: LevelSolved	
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 Methods	**Method Description**
+ next_level()	Navigates the user to the next level.
+ level_select()	Navigates the user to the Level Select menu.
+ restart_level()	Resets and resumes the current level.
+ main_menu()	Navigates the user to the Main Menu.

Class: PauseMenu	
Class Description: This class is used to draw and process inputs for the Pause menu.	
Class Methods	**Method Description**
+ resume_game()	Returns to the game, closing the Pause menu.
+ instructions()	Navigates to the instructions for the particular level.
+ restart_level()	Resets and resumes the current level.
+ exit_menu()	Navigates the user to the Main Menu, closing the current level.

Class: Instructions
Class Description:

Class: Instructions	
Class Description: This class is used to show the instructions for the current level.	
Class Members	**Member Description**
+ title	Title of the instructions.
+ body	The instructions which contain further details about the level.
Class Methods	**Method Description**
+ exit()	Exit the menu.

Class: IncorrectSolution	
Class Description: This class is used to tell the player that their solution was incorrect when the player tests their solution.	
Class Methods	**Method Description**
+ resume()	Resumes the level.

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: Vector2D	Stores the player's current position in the level.

Class: Player	
+ gate_selected: int	Store's the gate ID which the player is trying to place.
+ tool_selected: int	Store's the tool ID which controls if the player is trying to connect or disconnect gates.

Class Methods	Method Description
+ move(delta: Vector2D)	Moves the player within a level. The player is moved by the specified coordinates, not to them.
+ remove_gate(gate: Gate)	Removes a gate from the level.
+ place_gate(gate: Gate)	Places a gate within the level. The gate is placed where the player is looking at, aligned to the grid.
+ select_output(gate: Gate)	Stores the gate the player is looking at, so the gate can be connected to an input of another gate.
+ select_input(gate: Gate)	Connects the selected output to the selected input and places wire between them on the grid.
+ disconnect_input(gate: Gate)	Disconnects an input from a gate and removes the wire between the output gate and input gate.

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

Class: Gate	
	active, and False indicates that the state is inactive.
+ output: Boolean	Stores the current state of the output, 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.

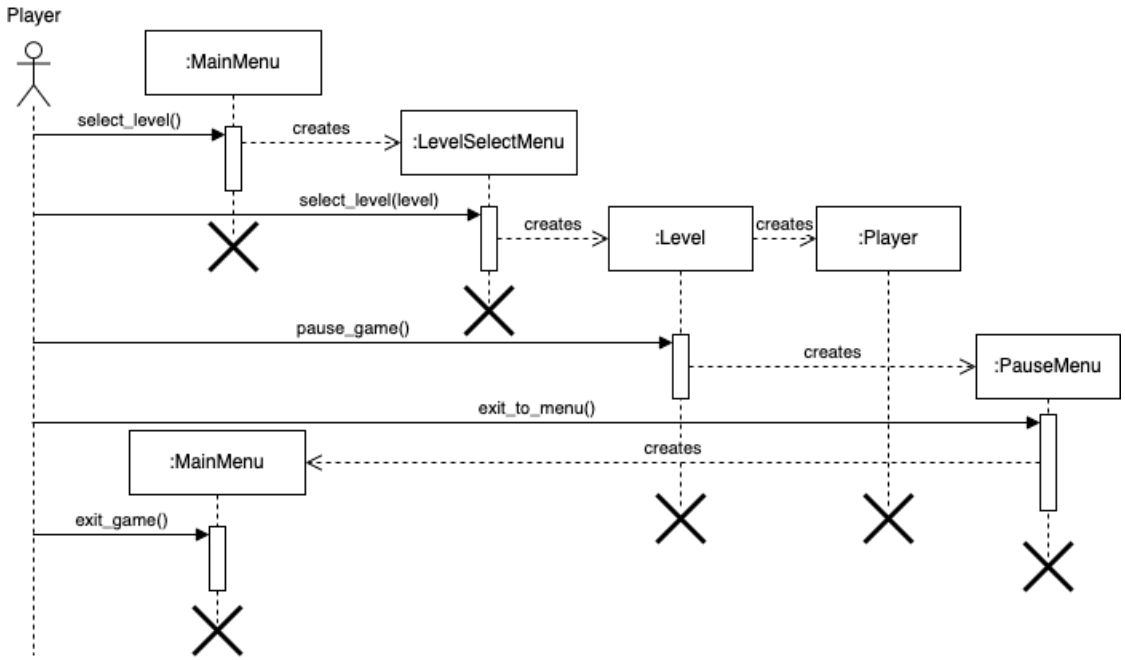
Class: Wire	
Class Description:	
This class is used to draw and update logic for wires placed within a level.	
Class Members	**Member Description**
+ input: Gate	Stores the input gate that this piece of wire receives a signal from.
+ output: Gate	Stores the output gate that this piece of wire sends a signal to.
+ path: List<Point2D>	Stores the path of the wire along the grid.

4.3 Sequence Diagrams

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

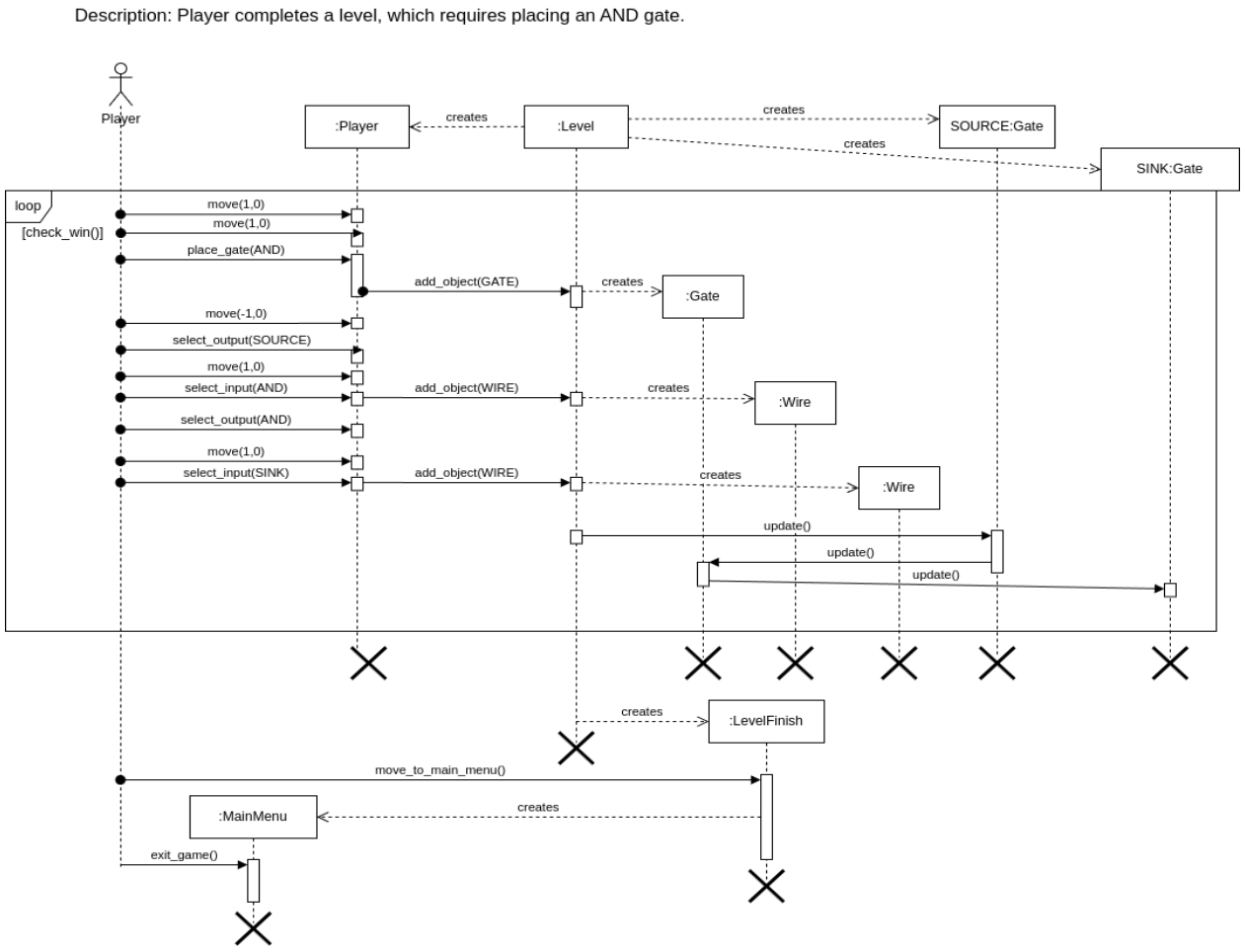
the game, and presses the “Main Menu” button to return to the Main Menu. The player then presses the “Exit” button to quit to the desktop.

Figure 4.3.1 – Sequence Diagram 1



The player is in the AND Gate tutorial level and must use the navigation keys to move around the environment. The player has to place an AND gate, then switch from the AND gate to the wire tool in the hotbar. The player can then use the select output tool on the SOURCE block. Next the player must select the AND gate with the input tool and then select the AND gate with the output wire tool. The player can then walk over to the SINK block and select it with the input wire tool. Finally, the player can press “SPACE” to test their solution.

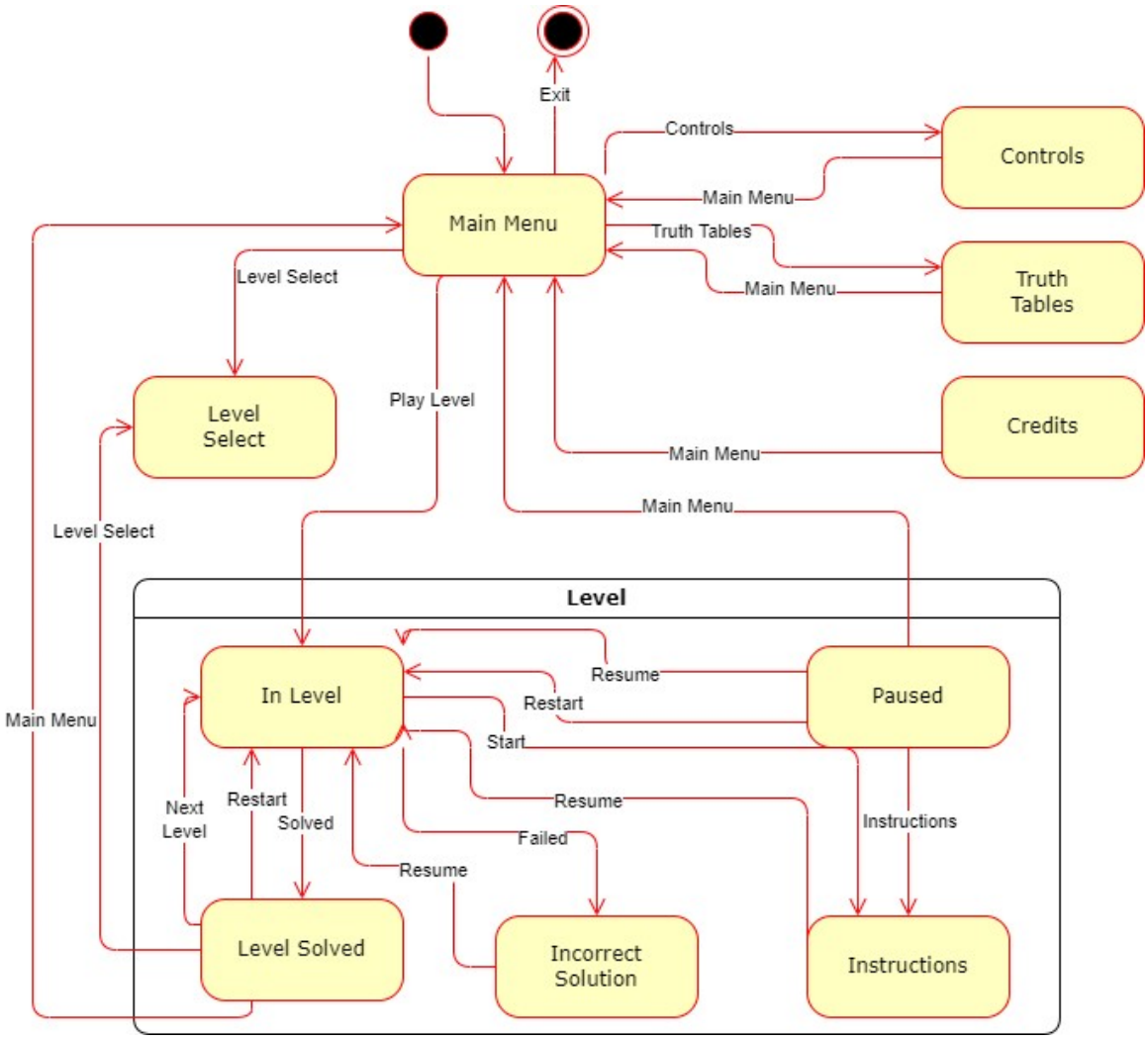
Figure 4.3.2 – Sequence Diagram 2



4.4 State Diagrams

This state diagram models the primary navigation of the game, including playing and completing a level, pausing and returning to the game, and selecting a level. The navigation features the Main Menu, the Level Select Menu, the Pause and Level Solved Menus, the Truth Table Menu, Control Menu, Instruction Menus, and Credits Menu.

In this state diagram, there are ten states: the ‘In Level’ state, the terminal state, and the six menus. The game is initially in the ‘Main Menu’ state. The ‘Controls’, ‘Truth Tables', and ‘Credits’ states are each self-contained, and include transitions back to the ‘Main Menu’ state. The ‘Level Select’ state contains a transition from the Main Menu state, and a transition to the ‘In Level’ state. This state includes transitions to the ‘Paused’ state, which can transition to the ‘Main Menu’, ‘Instructions’ or ‘Level Select’ states, or return to the ‘In Level’ state. The ‘Level Solved’ state can transition to the ‘Level Select’ or ‘Main Menu’ states, or continue to the next level via the ‘In Level’ state. Lastly, the ‘Main Manu’ state can exit the game, transitioning to the terminal state.



5 Prototype

The prototype demonstrates the navigation, controls, UI, and showcases an example level progression. The prototype includes navigation through the Main Menu, the Level Select Menu, the Pause Menu, the Level Solved Menu, the Credits Menu, the Controls Menu, the Truth Tables Menu, and the Credits Menu. The prototype demonstrates the controls for movement, placing gates, connecting gates using wires, and testing levels. The prototype showcases the UI for the aforementioned menus, as well as the hotbar used to select gates and the cursors that indicate where the gates and wires are placed. The prototype includes a progression from simple one-gate puzzles, to more complex logic elements like decoders, multiplexers, adders, and memory cells.

5.1 How to Run Prototype

5.1.1 Windows

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

5.1.2 Linux

1. Install Flatpak for your distribution: https://flathub.org/setup
2. 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 application and arrives at the main menu:

Figure 5.2.1 – Main Menu

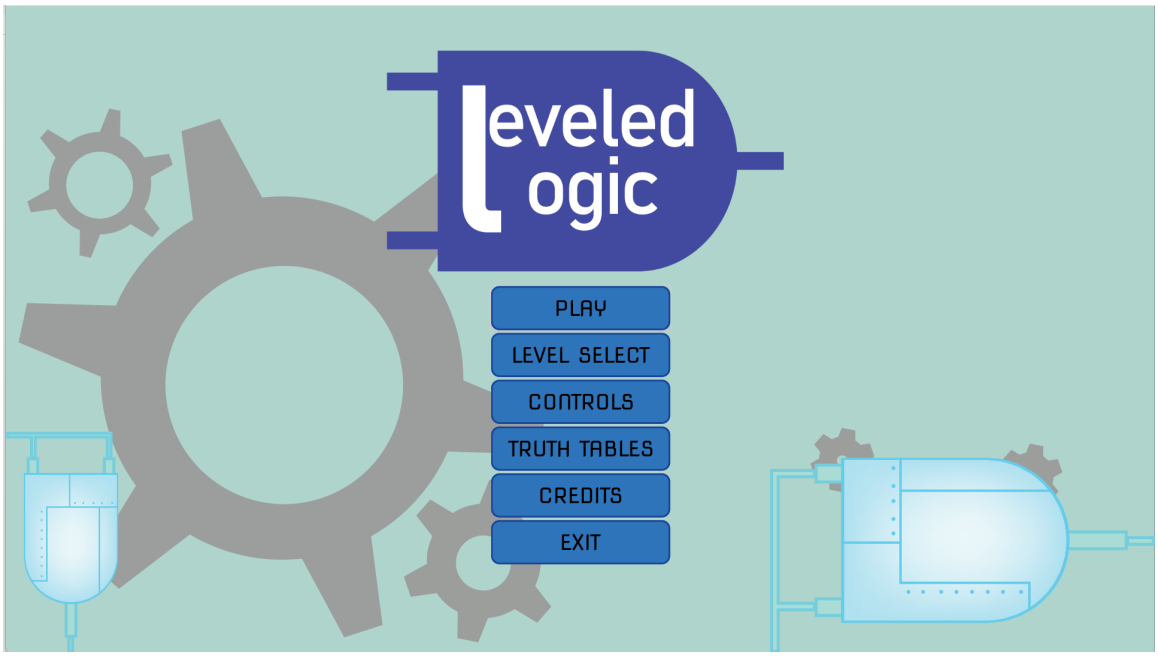


Figure 5.2.2 – Level Select Menu

The Player selects “LEVEL SELECT”, bringing them to the level select menu:

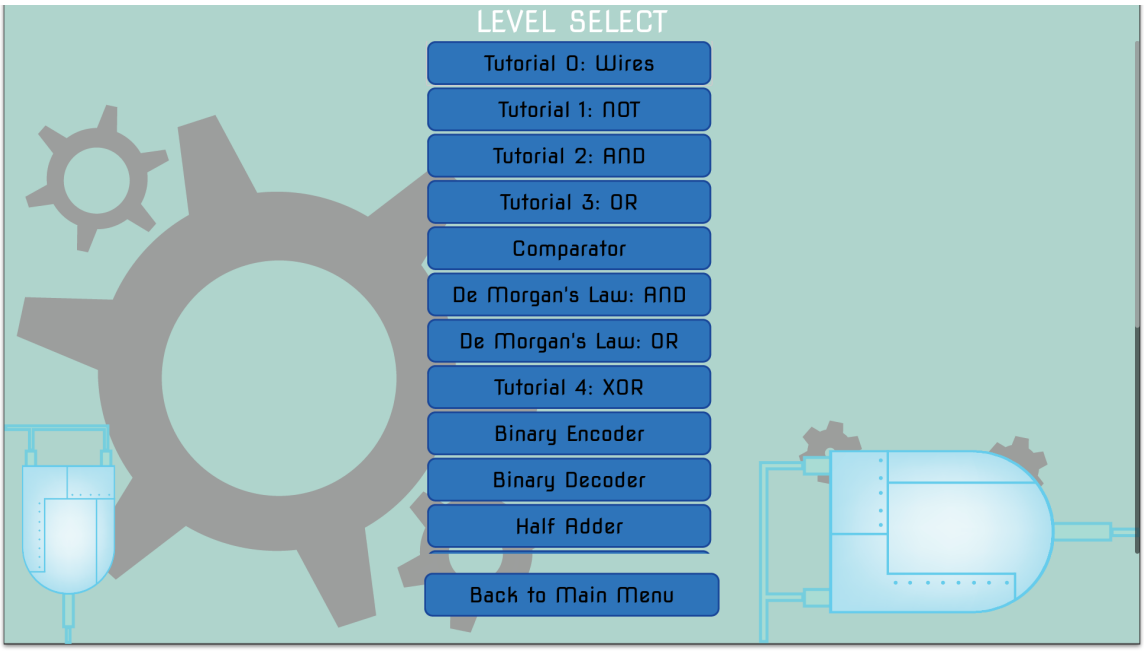


Figure 5.2.3 – De Morgan's Law: AND Level Instructions

The Player selects "De Morgan's Law: AND", bringing them to a level:

Figure 5.2.4 – De Morgan's Law: AND Level

The Player presses "EXIT" on the instructions:

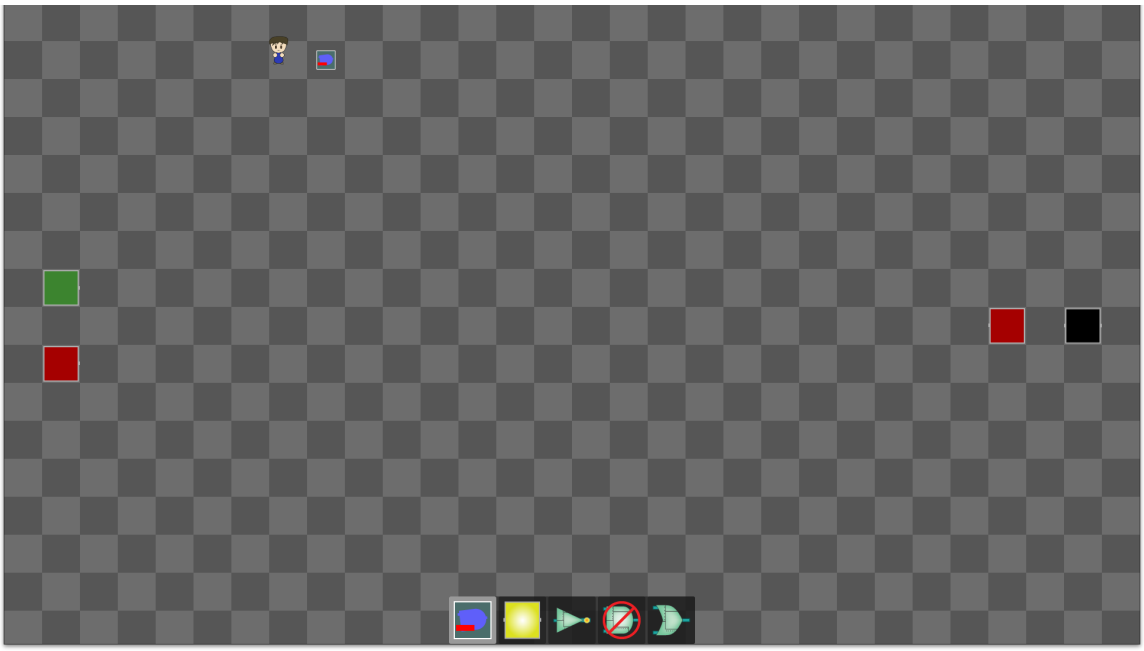


Figure 5.2.5 – De Morgan's Law: AND Level

The Player selects gates from the hotbar and places them on the grid:

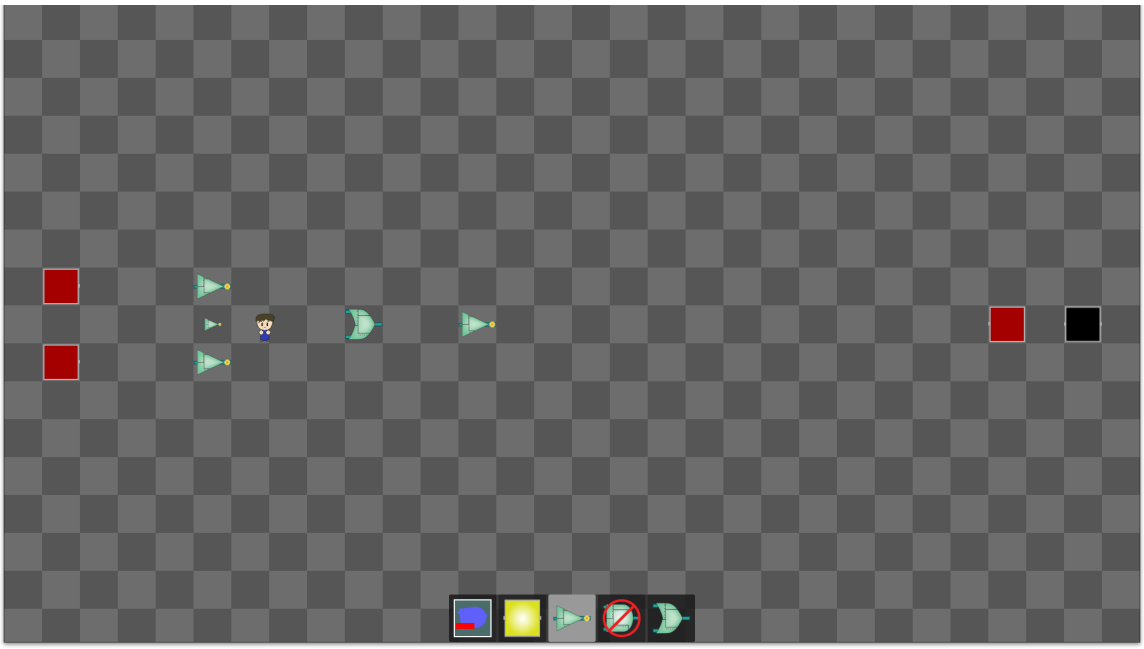


Figure 5.2.6 – De Morgan's Law: AND Level

The Player switches hotbar to the Wire Tool Hotbar:

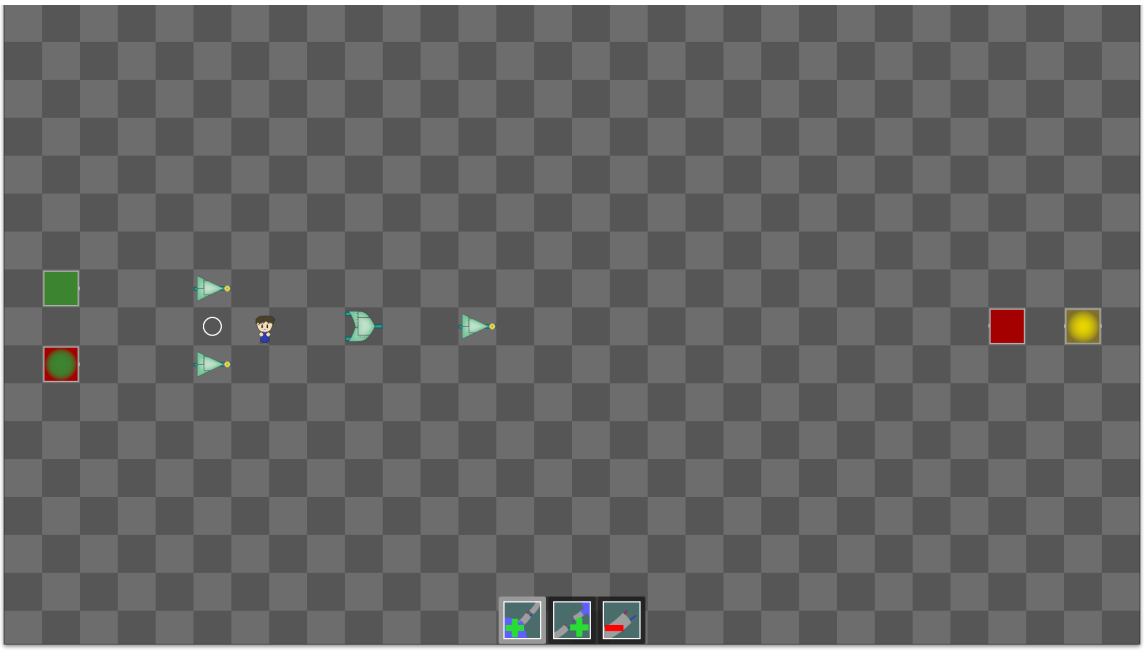


Figure 5.2.7 – De Morgan's Law: AND Level

The Player connects gates together:

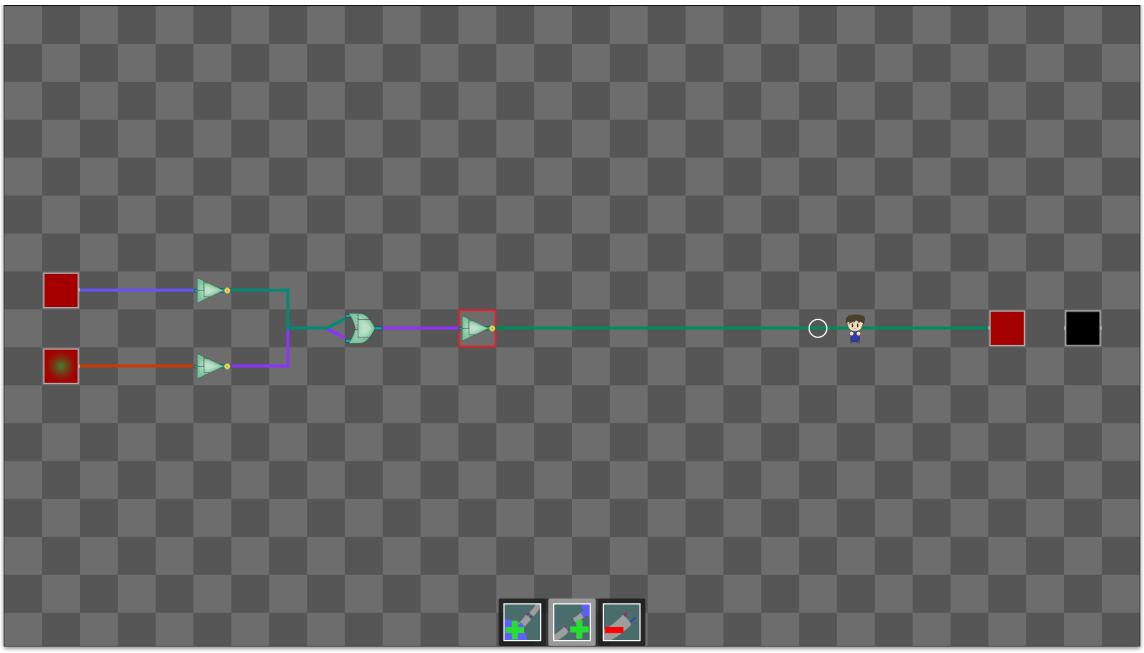


Figure 5.2.8 – De Morgan's Law: AND Level

The Player presses "Space" key to test their solution:

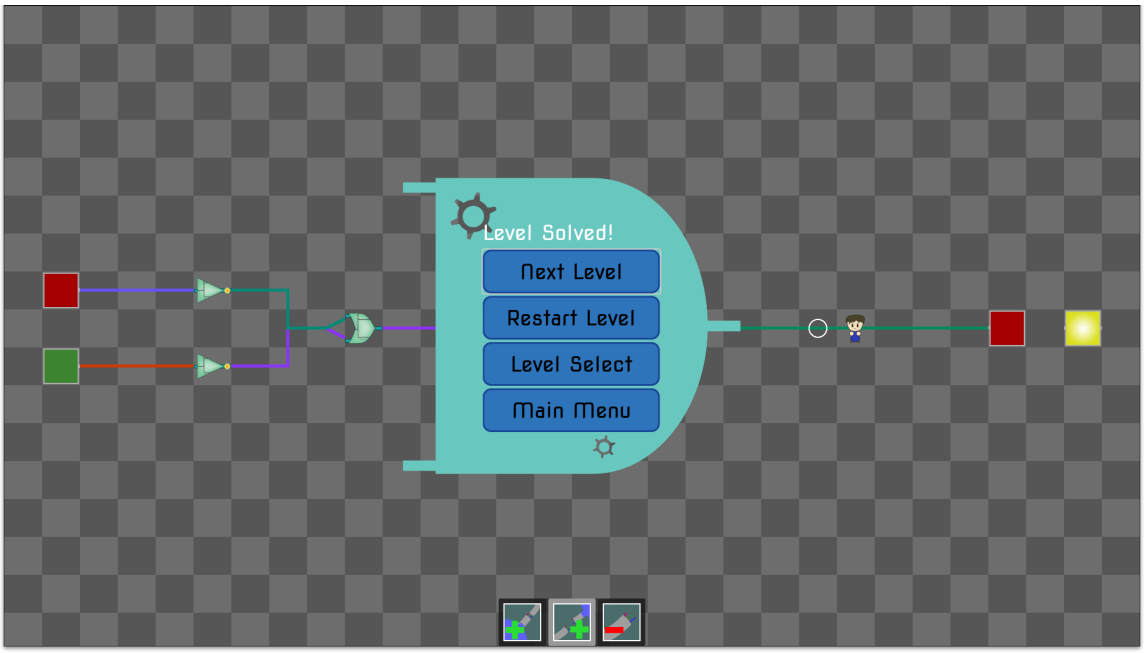
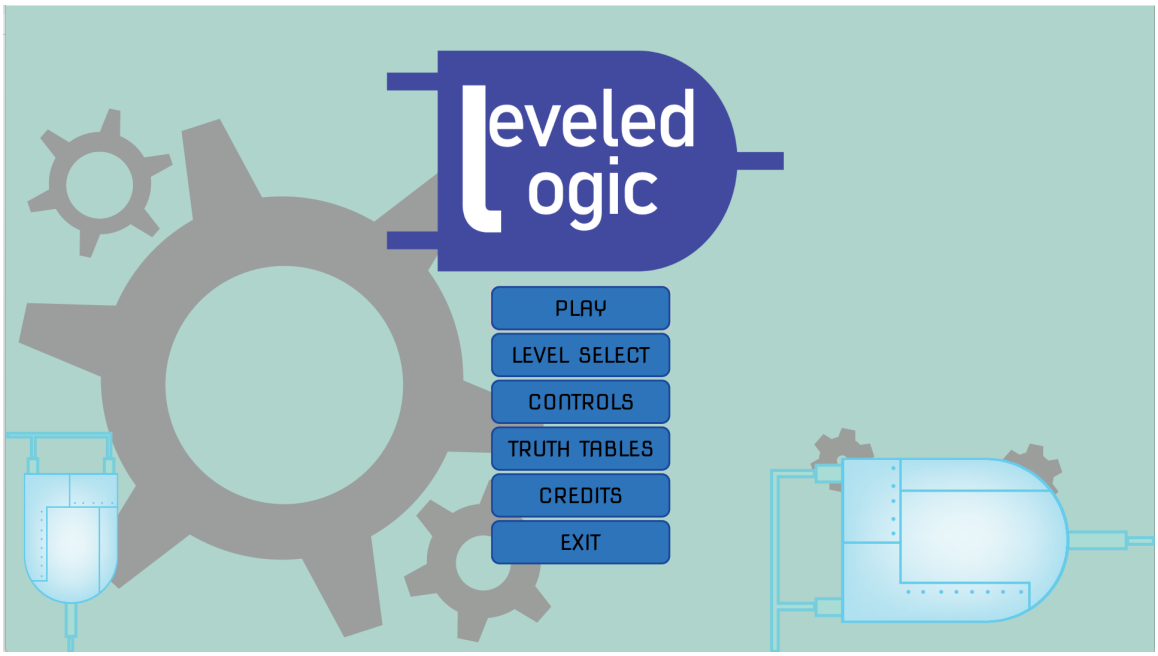


Figure 5.2.9 – Main Menu

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



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

6 References

[1] 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).

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

[3] 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

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

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

[6] Bucks County Community College, "Truth Tables," [Online]. Available: https://www.bucks.edu/media/bcccmedialibrary/tutoring/documents/math/Truth-Tables.pdf. Accessed on Dec. 08, 2023.

[7] Entertainment Software Rating Board, "Does an E or E10+ rating mean a game or app is directed to children for purposes of COPPA?," [Online]. Available: https://www.esrb.org/privacy-certified-blog/does-an-e-or-e10-rating-mean-a-game-or-app-is-directed-to-children-for-purposes-of-coppa/. Accessed on Dec. 08, 2023.

[8] Unity Technologies, "Top-down games," Unity Forum, [Online]. Available: https://forum.unity.com/threads/top-down-games.120499/. Accessed on Dec. 08, 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.