Liam Jerremy V. Tolentino

ENGL 1102

Dr. William Lawson

11 December 2023

# On the Turing Completeness of *Portal 2*

## Introduction

In 1936, Alan Turing proposed the possibility of inventing a machine that can calculate any computable number or sequence of numbers that another machine can do just by taking in the description of the other machine as an input, and can even simulate itself if given its own description as input (Turing, 241). This machine came to be known as a "universal Turing machine", as it is capable of computing anything that another computing machine can do. It is from this concept that the term "Turing complete" originated, a term used for systems such as programming languages and operating systems that "can be used for any algorithm, regardless of complexity, to find a solution" (Yaga, 54). Turing complete systems are the reason why modern computers and devices are capable of running the seemingly endless number of applications and programs that are used today. While Turing completeness is usually applied to more practical systems, such as electronics and programming languages, some have proven the Turing completeness of some arbitrary systems that would never be practically used for computing in the same way that a laptop or smartphone would be used. For example, Alex Churchill proved that the fantasy card game *Magic: The Gathering* is Turing complete by showing a way that the game can theoretically be played to simulate the functions of a universal Turing machine which can therefore be used to simulate any other Turing complete system (Churchill et al., 2). In a similar way, Hull and Zakharevich prove that origami is Turing complete by showing how folded

crease patterns can simulate another system that has already been proven to be Turing complete (Hull and Zakharevich, 15). While these systems could never be practically used to do any actual computing work, they provide deeper insight into the possibilities of computing systems and can help further the development of more practical systems such as quantum computers or any other potentially groundbreaking discovery in the future. It is for this reason that this project aims to prove that the 2011 video game *Portal 2*, or at least the puzzle elements in the game, are Turing complete.

**An Explanation of Turing Machines**

In order to understand how to prove the Turing completeness of *Portal 2*, it is important to first explain how Turing machines work. The Turing machine, or computing machine as it was called in Turing's paper, is not a physical machine that can be built in the real world, but rather a hypothetical machine that when simulated can compute a number or sequence of numbers it is programmed to calculate. Turing describes the machine in the following passage:

> We may compare a man in the process of computing a real number to ;i
> machine which is only capable of a finite number of conditions q 1: q 2 . .... q I;
> which will be called " m-configurations ". The machine is supplied with a
> " t a p e " (the analogue of paper) running through it, and divided into
> sections (called "squares") each capable of bearing a "symbol". At
> any moment there is just one square, say the r-th, bearing the symbol <2>(r)
> which is "in the machine". We may call this square the "scanned
> square ". The symbol on the scanned square may be called the " scanned
> symbol". The "scanned symbol" is the only one of which the machine
> is, so to speak, "directly aware". However, by altering its m-configu-

> ration the machine can effectively remember some of the symbols which
> it has "seen" (scanned) previously. The possible behaviour of the
> machine at any moment is determined by the ra-configuration q n and the
> scanned symbol <S (r). This pair q n , © (r) will be called the " configuration":
> thus the configuration determines the possible behaviour of the machine.
> In some of the configurations in which the scanned square is blank (i.e.
> bears no symbol) the machine writes down a new symbol on the scanned
> square: in other configurations it erases the scanned symbol. The
> machine may also change the square which is being scanned, but only by
> shifting it one place to right or left. In addition to any of these operations
> the m-configuration may be changed. Some of the symbols written down will form the
> sequence of figures which is the decimal of the real number
> which is being computed. The others are just rough notes to "assist the
> memory ". It will only be these rough notes which will be liable to erasure. (231-232)

In other words, the machine is given an infinitely long sheet of paper, or “tape” as Turing calls it, which is divided into equally sized squares on which the machine can read or write a symbol. The machine can only read one square at a time, and depending on what symbol is currently being read and in which configuration the machine exists, the machine will then write a particular symbol onto the current square and then move to the next square either to the right or the left of the current square. Leavitt’s book explains this with a woman in a factory as an analogy, but the following explanation will make a few changes to hopefully make this easier for the reader to understand (Leavitt, 60-66). Imagine a woman tasked with solving a specific math problem. She is given an infinitely long sheet of paper to do her work, but she is only allowed to

see a small square section of the paper at a time and each section can only have a single symbol on it. The woman is also given a book and each page of the book has a table of rules with every possible symbol that can appear on the paper listed and tell her what symbol to write on the current square of paper, which square to read next (the one to the right or to the left of the one she is reading), and which page of the book she should turn to next which will have a different table of rules to follow for the next square. This means that even if the current square shows the same symbol she saw before, she might do a different series of steps if she is on a different page of the book. The infinitely long paper of squares is the tape of the Turing machine, the woman reading and writing on the squares is the head of the machine, and the pages of the book are the configurations of the machine. Depending on what is written on the pages of the book and how the pages lead to one another, one can give the woman a book to calculate a specific number or sequence of numbers that they need to compute. The problem here is that sometimes it is not possible to give the woman a different book as the book may be hardwired into the system. This problem is addressed with a special Turing machine called a universal Turing machine.

The introduction section of this paper briefly explained the universal Turing machine, and now it can be explained further with the woman in a factory analogy. Suppose this time, the woman is given a special book that allows her to do the same computations that another book allows her to do. Rather than just giving her a new book to do the computation, one can translate the new book into a long string of symbols and put it into the infinitely long paper. The woman can then use her current book to read the translated symbols of the new book and simulate what the new book tells her to do just by following the steps of the special book. This special book allows the whole system to be Turing complete as it is possible to then give the system new instructions by just writing symbols on the paper that describe another machine's instructions

rather than completely swapping out the instructions. What makes this special book even more special is that one can also translate that same book onto the tape, or even another special book that's Turing complete, and repeat that infinitely, meaning that the woman and the special book can simulate another special book or even the full operations of a Turing complete operating system running a program simulating the woman and the special book.

**Examples of Turing Machines**

To help further explain the functions of the Turing machine, this section will explore some examples. Some of these will be visited again in later sections as they will be used to demonstrate some of the findings of the experiment.

Firstly, we will shift away from the woman in a factory analogy to explain state diagrams as they provide a more visual way to explain the Turing machine examples. Now, imagine instead of a book, the woman is given a map. The map has several points each corresponding to a page of the book, or a configuration of the Turing machine. We will call these points states, and for the rest of the paper, the configurations of the Turing machine will also be called states to maintain consistency. Each of these states have arrows pointing away from them and towards the next state to visit depending on the instruction. These arrows each have labels that tell the woman which symbol she needs to read in order to follow the arrow as well as what new symbol to write and which direction to move the tape once she finds the arrow she needs to follow. These arrows correspond to the rules listed in each of the pages of the book. Now, instead of flipping through pages in a book, the woman follows this map and depending on where she is and where she is going on the map, she will have specific instructions on what to write on the tape.

Figure 1.1 shows the state diagram for a very basic machine that Turing himself described in his paper (Turing, 233). This machine simply writes alternating zeros and ones on the tape with empty spaces in between. Notice that this state diagram layout represents the instructions of the machine in one page rather than multiple pages in a book. While the book analogy better represents the limited visibility given to the machine, the state diagram allows one to visualize how the different states of the machine connect to one another. Also notice that a blank square still counts as a symbol. This will be important later when deciding how to encode symbols as the blank symbol is still important to count.

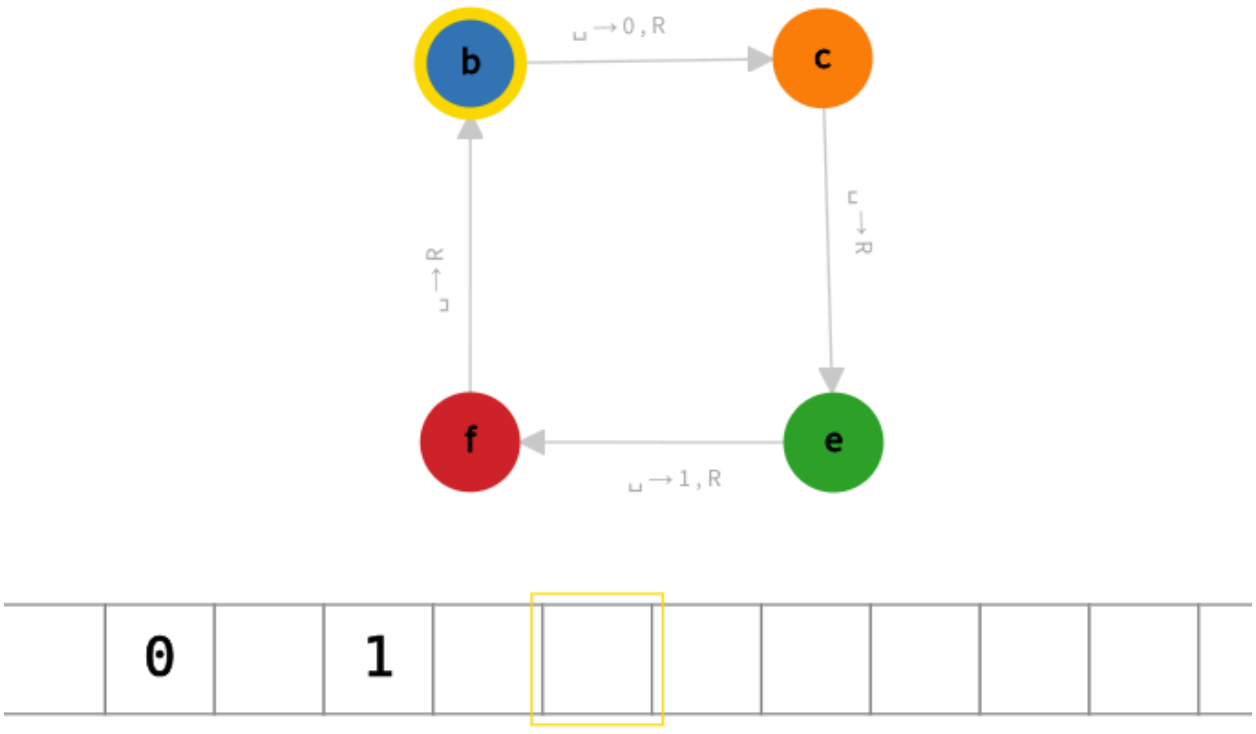
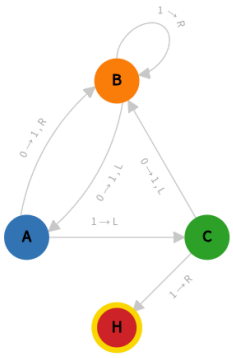



Fig. 1.1. The state diagram for Turing's repeating symbols machine after four cycles

The next example is what is known as a 3-state busy beaver (fig. 1.2). For the purposes of this paper, it is not important to know what a busy beaver is, but the state diagram does demonstrate many of the important functions of a Turing machine. Firstly, notice the state H and how there are no arrows pointing away from it. This is what is known as a halting state and it marks when a Turing machine is finished with its computations. Unlike the previous example which loops forever, this machine eventually reaches a point where it can stop. It is not always necessary to show a halting state for a machine to stop, as seen in the next example, but it is still important to consider it when constructing a Turing machine that needs to stop at some point. Also, notice how this example uses only two symbols and no blank symbols as the blank is represented by zero. This goes back to what was stated earlier about how the blank symbol must be counted when constructing a Turing machine. Finally, notice how this machine occasionally moves to the left unlike the previous example that only moved to the right. This ability to return to a previous position is important as it allows the creation of a machine that needs to keep track of what it has already done.



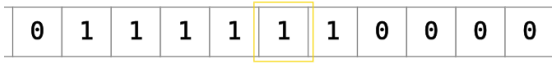




Figure 1.2 A 3-state, 2-symbol busy beaver Turing machine after completing its operation

The last example is an example of a universal Turing machine, or in terms of the woman in a factory analogy, this is the special book that allows her to simulate other books with just an input from the tape (see fig. 1.3). More specifically, this is an implementation of a universal Turing machine shown in Rogozhin's paper proving the existence of certain small universal Turing machines (Rogozhin, 226). This machine is notated as UTM(7,4) with the seven referring to the seven states/configurations of the machine and the four referring to the four valid symbols it can read and write. Churchill's paper proved the Turing completeness of *Magic: The Gathering* by simulating a UTM(2,18) which was also proven in the same paper by Rogozhin (Churchill et al., 2; Rogozhin, 236). For reasons that will be stated in a later section, we have chosen to simulate UTM(7,4) to prove the Turing completeness of *Portal 2*.

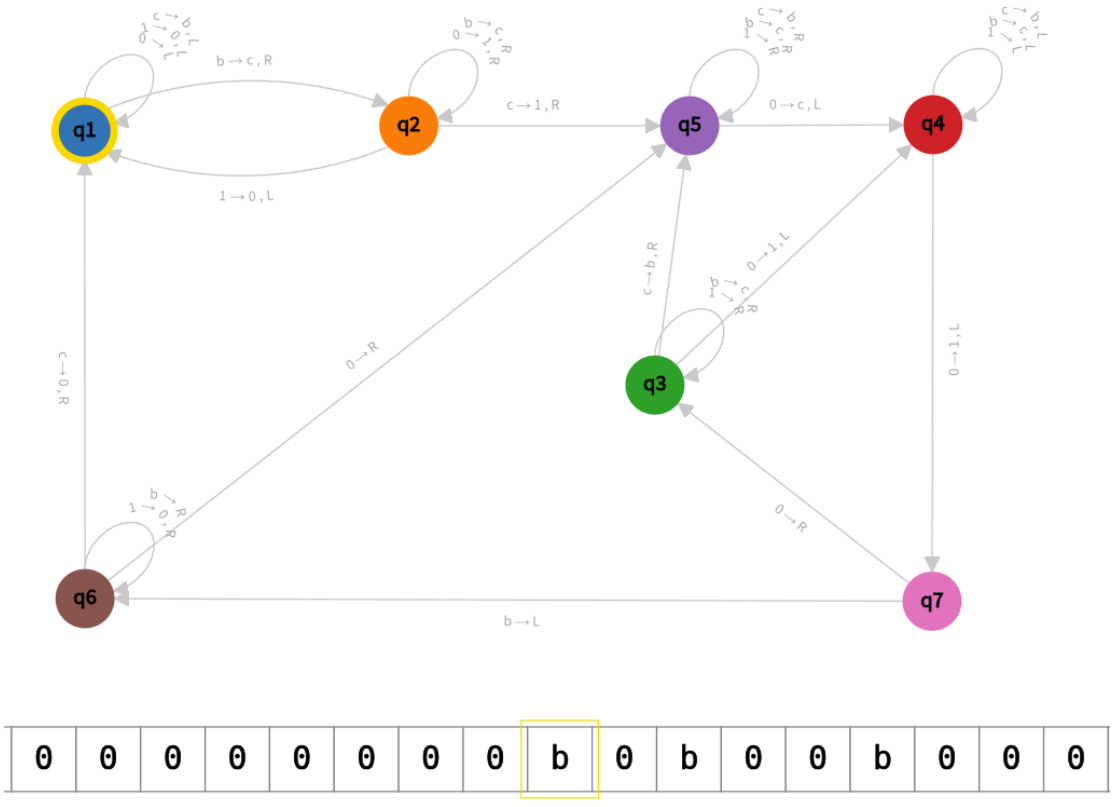



Fig. 1.3. State diagram for UTM(7,4) before starting its operations

Notice on the diagram that there is no circle representing the halting state. Instead, q7 is missing two arrows pointing away from it that would normally point to a halting state. As stated before, the halting state does not need to be shown, but it is important to remember it when constructing the machine so that it can halt. Also, notice how the tape already has symbols written on it before the machine has even started. Normally, these symbols are supposed to represent the functions of another Turing machine, but the actual process of encoding a Turing machine is too complex to include in this paper and is not necessary for the overall goal.

**The Mechanics of *Portal 2***

Now that the functions of the Turing machine have been explained, it is now time to explain how the game *Portal 2* can be used to theoretically simulate the computations of these machines. The game is mostly known for the two built-in story campaigns where players solve puzzles to progress through a story; however, there is also a third option in the game menu called "Community Test Chambers" in which players can play puzzles that others have created as well as create their own puzzles through an in-game "puzzle creator" (see fig. 2.1). This puzzle creator is similar to features in other games such as "creative mode" from *Minecraft* or the "Forge" gamemode from *Halo 3* as they allow for users to freely manipulate the environments in the game without the use of external tools or applications. It will be through the puzzle creator that the components of the Turing machine will be simulated as this allows the experiment to be conducted entirely within the game and will allow us to prove that *Portal 2* is Turing complete.

It is important to note that a lot of the community-made puzzles and even the built-in story campaigns mentioned before were not created with the in-game puzzle editor, but rather an external tool known as the Hammer Editor which is also used to create maps for other games running on the Source engine. For the purposes of this project, the Hammer Editor will not be

used to prove the Turing completeness of *Portal 2* because for one, it is an external application and would defeat the purpose of having the experiment take place in-game, and also, even if a Turing machine were to be constructed in this Hammer Editor, it would not prove that *Portal 2* is Turing complete, but rather that the Source engine, the engine on which the game was built, is Turing complete.

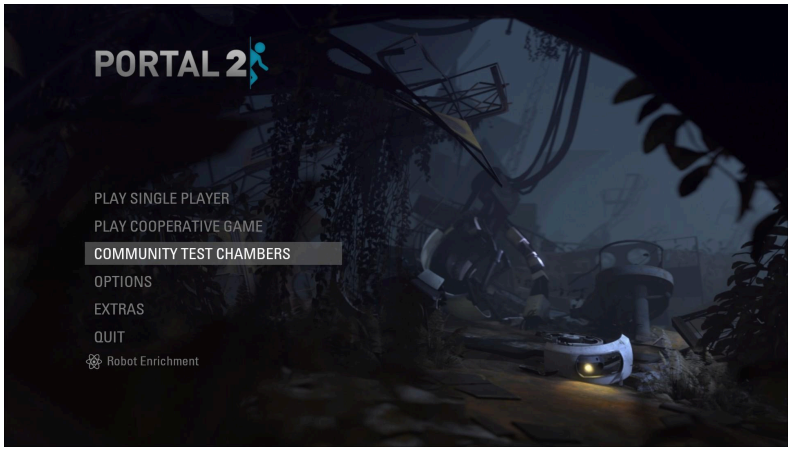



Fig. 2.1. *Portal 2* main menu with the option for Community Test Chambers selected

Another important detail to address is that the in-game puzzle editor has strict restrictions on the size of puzzles and the total number of puzzle elements used. This makes it impossible to actually construct the design that will be presented later in the game with only the in-game features. This will not necessarily disprove the Turing completeness of *Portal 2* as the design still follows the rules of the in-game puzzle creator and only breaks the size and item count limitations, so it is still theoretically possible, with a longer timeframe, to construct a more compact Turing complete puzzle in the game using concepts from the proposed design. Also, one can argue that the size limitation does not matter for Turing completeness as the Turing machine itself is infinitely large so none of the "Turing complete" systems created in the real world can truly simulate the full functions of the machine. However, with the limitations in mind, it is equally important to note that the experiment will not fully confirm the Turing completeness of

*Portal 2*, but it does show that the elements accessible in the built-in puzzle creator can be used to simulate a Turing complete system. This argument will be discussed in a later section of the paper, but for now, allow us to explain how the experiment will be conducted in the game.

*Portal 2*'s in-game puzzle creator allows the user to manipulate environment surfaces and to attach and modify puzzle elements on these surfaces. While the game provides thirty-two different elements to use, most of the designs shown will only use the laser emitter, laser catcher, and laser relay, with the weighted cube and button used occasionally for user input (see fig. 2.2). The laser-based elements are ideal for this experiment as they are consistent and predictable and they react almost instantaneously allowing us to build circuits that can be timed easily. MystX1 took a similar approach with their video about the 8-bit adder circuit they made in the game which reacts quickly to user input (MystX1, 0:35-0:50). This responsiveness makes it easier to quickly test certain designs in-game as they will perform their functions very quickly and since these elements are always attached to a surface, there will be no need for moving parts which may cause inconsistencies in the designs. However, as will be seen later in the testing section, there is a problem with *Portal 2* lasers that may require moving parts to solve.

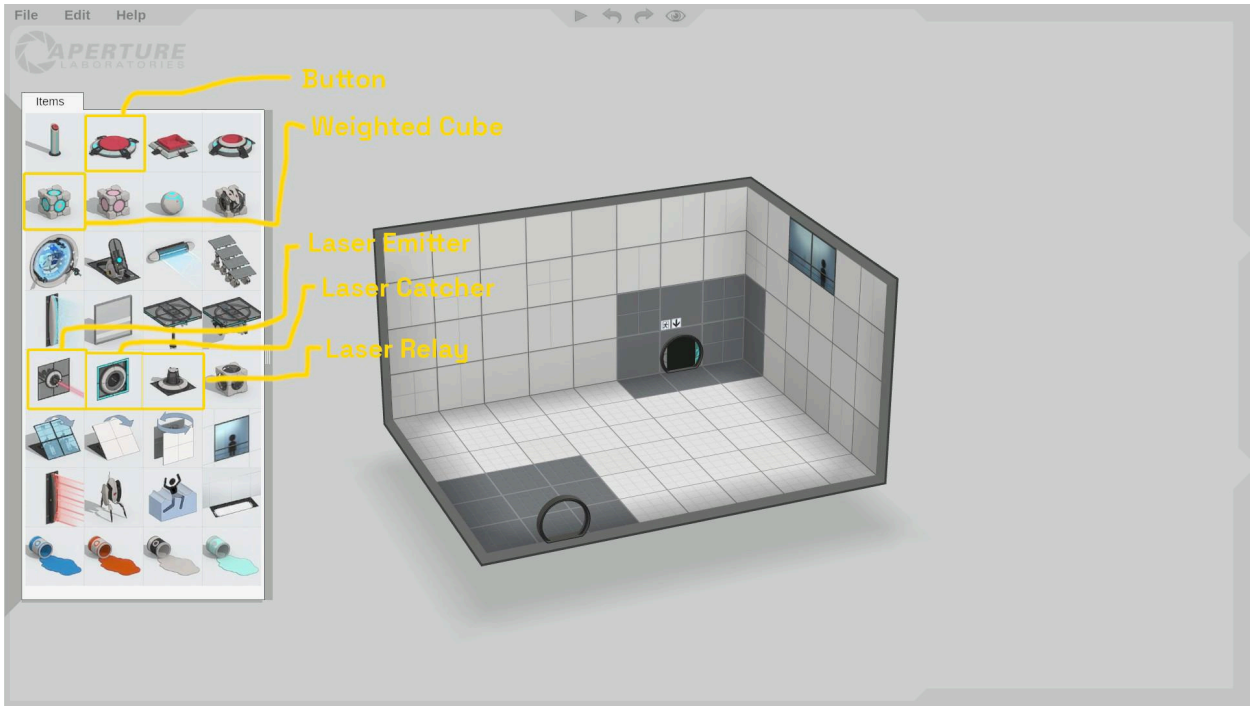



Fig. 2.2. A screenshot of a newly created puzzle in the in-game puzzle creator with the relevant elements highlighted and labeled

By clicking on an element in the puzzle creator with the right mouse button, a menu will pop up with a list of options (see fig. 2.3). Clicking the option labeled “Connect to…” will allow the user to then click on another puzzle element, and as long as one element is an input element and the other is an output element, the game will then create a line of blue dots to visually show that they are connected. This line of dots is referred to as an “antline” in-game. Input elements, such as the button, laser catcher, and laser relay will activate any output elements, such as laser emitters, that are connected to them via an antline if they themselves are activated. This means that a laser emitter connected to a button will turn on its laser beam once the button has been activated, or in the case of the laser emitter in figure 2.3, it will turn off its laser beam if the “start enabled” option has been selected, acting as a signal inverter to any input connected to it. Buttons are activated if there is a player or a weighted cube placed on top of them, allowing for

direct user interaction with contraptions. The laser catchers and laser relays only activate if hit by a laser beam from a laser emitter which means that it is possible to construct logic circuits using these elements that can connect and interact with each other by having laser emitters from certain circuits point to the input elements of other circuits. This would allow each component of the Turing machine to be constructed individually as long as they are designed to have outputs and inputs that can directly interact with the other components, similar to how real life electronics can have multiple individual components that can connect to each other and be interchanged with other compatible circuits.

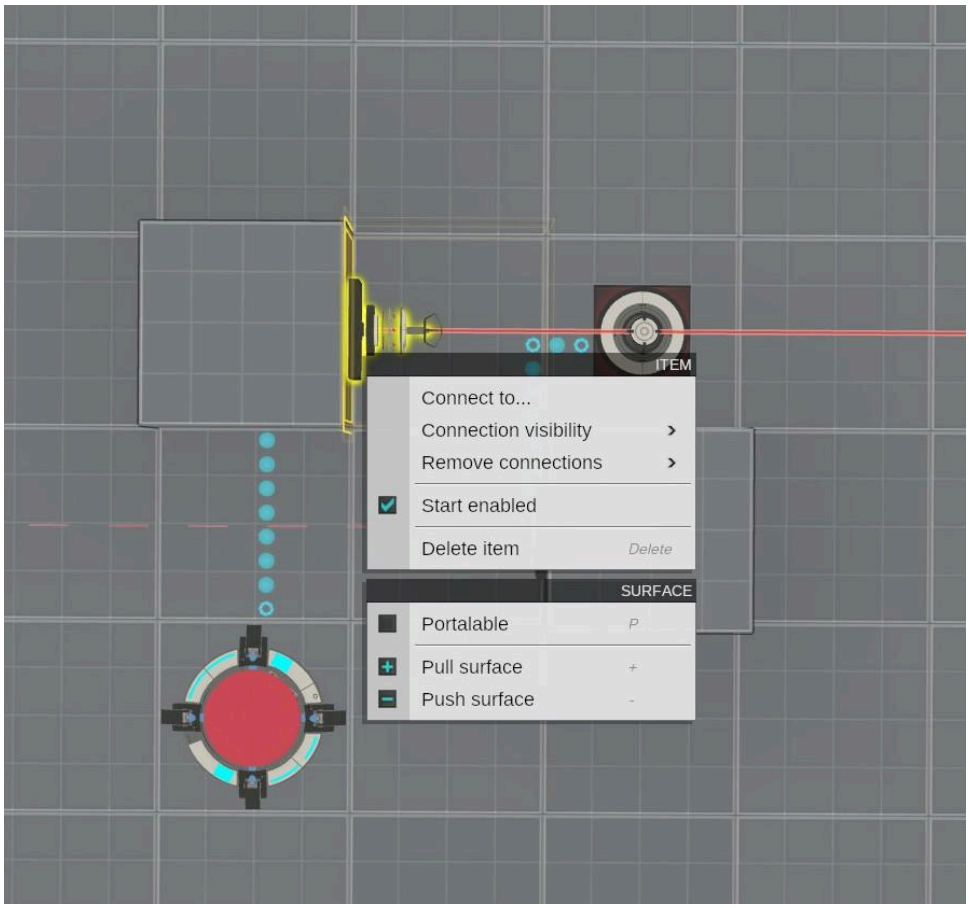



Fig. 2.3. A screenshot from the in-game puzzle creator showing an example of how puzzle elements connect to each other as well as the menu for editing elements

Before explaining how these puzzle elements will be used to construct the components of the Turing machine, there are a few more game mechanics to explain. Firstly, puzzle elements do not have to be connected to only one other element as they can be connected to other ones which can lead to some useful interactions. If an input element, such as a button, was connected to multiple output elements, all of the connected outputs would simultaneously activate once the input is activated. If an output element had connections to multiple input elements, the output element would only activate if all of its inputs were activated at the same time. The laser relay has a special property in that the laser beam that activates it will also pass through which means that a single laser emitter can activate multiple relays at once and therefore multiple circuits that use a relay input can receive the signal from an output of one circuit. One last detail to point out, especially those who wish to replicate this experiment, is that elements such as the laser emitter and laser relay can be adjusted with a small white square while selected in the puzzle creator which can be dragged to move them to one of five positions on the tile on which they are placed. This is important because in order for a laser emitter to aim at a laser relay, the emitter needs to be placed at the bottom of the tile. If it is left in its default position in the middle of the tile, the laser will simply pass over the top of any relays in front of it and not activate any of them.

**Implementation of the Turing machine's components**

As mentioned earlier, the limitations built into the puzzle creator in *Portal 2* make it difficult to construct a fully functioning Turing machine in the game without reaching the object count or room size limit, or experiencing game crashes due to the large number of objects that the game needs to calculate. It is for this reason that the only components of the Turing machine that were created in the game for this experiment were simple logic circuits that could be connected together to form the full components of this machine. The fully designed components

are therefore represented as diagrams that show exactly how these circuits can be combined to form functioning components of a Turing machine. A *Portal 2* puzzle designer with much more experience and knowledge with the puzzle creator tool can then use the diagrams to create a compact and efficient design that can run in the game to demonstrate the functions of the machine, but for now, the design remains on the drawing board. These limitations and setbacks do not outright disprove the Turing completeness of *Portal 2* as the papers from Churchill and Hull still manage to prove the Turing completeness of their respective systems without physically implementing the designs (Churchill et al., 4; Hull and Zakharevich, 11). It can be argued that since the design does not demonstrate how a universal Turing machine can be constructed in-game with all the rules and limitations of *Portal 2*'s built-in mechanics, the game itself is not Turing complete; however, since the only limitations are the size and object count and the design still adheres to the tools and elements provided by the game, it is fair to say that the puzzle elements themselves form a Turing complete system.

The basic logic circuits needed to build a Turing machine in *Portal 2* are the AND gate, OR gate, NOR gate, NOT gate, SR latch, and D latch with the last two being derived from the AND gate and OR gate. Figure 3.1 shows how the first four can be implemented in the puzzle editor as well as their respective schematic symbol shown on the right. Most of these were inspired by the video by Bungle, though only the OR gate follows his design exactly with the rest of them designed to also use laser elements (Bungle, 1:32). The NOR gate and NOT gate need to have their output laser set to "start enabled" similar to the example in figure 2.3 so that the output is inverted much like how their real life counterparts function. These designs do not need to be copied exactly as seen in the figure and the elements can be moved around as needed as long as the necessary connections remain the same. The AND gate accepts multiple inputs and

will activate its output if all inputs are activated. The OR gate also accepts multiple inputs but will activate its output if at least one input is activated. The NOR gate simply outputs the inverse of the OR gate meaning that its output is always activated unless one of its inputs is activated in which case the output turns off. The NOT gate takes only one input and simply outputs the inverse of the value, so if the input is off, then the output is on, and if the input is on, the output is off.

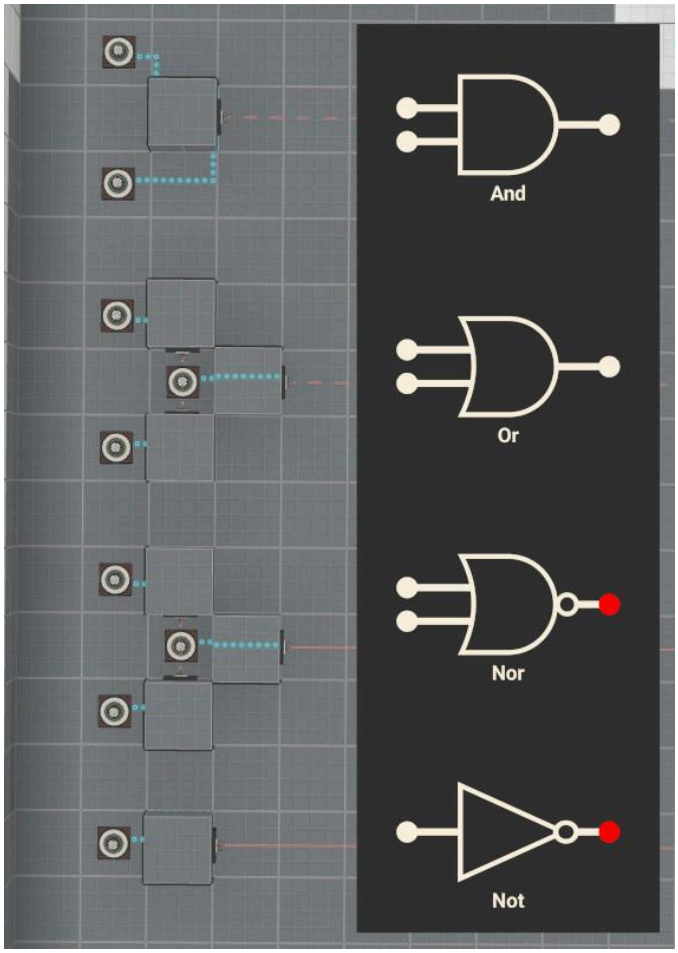



Fig. 3.1. Basic logic gate circuits in the *Portal 2* puzzle creator with their respective schematic symbols on the right

The SR latch and D latch are a bit different from the other circuits because they are constructed using other logic gates and also they can remain in the same state depending on

certain conditions which means that they can be used to store information. This will be useful for simulating the functions of the Turing machine's tape mechanism. As seen in figure 3.2, these circuits have names for their inputs and outputs to represent what they do. In both circuits, the node labeled 'Q' represents the output, or in other words, it represents the bit stored in the circuit with an activated output representing one and an off output representing zero. In the SR latch, the node labeled 'S' or "Set" in the case of the provided diagram will always set 'Q' on as soon as it is activated, so if 'S' received a laser while 'Q' was off, then 'Q' will turn on, and if it happens again while 'Q' was still on, then nothing will change. The node labeled 'R' or "Reset" does the exact opposite and turns 'Q' off every time it is activated, and will remain in that state until 'S' is activated again. It is important to note that because of the way *Portal 2* calculates lasers, any SR latch circuit in the level will immediately start pulsing on and off rapidly as soon as the level loads, but it can quickly be fixed by activating only one of the inputs of the SR latch. The D latch is a more advanced version of the SR latch, as seen by the presence of the SR latch in the circuit diagram, which means that it inherits this same problem; however, the D latch works differently. The D latch only changes its stored value if the node labeled 'E' is activated, otherwise it simply stays in whatever state it is currently in, no matter what is done to the other input node. Whenever 'E' is activated, the value of 'Q' is set to whatever state 'D' is in, so if 'D' happens to be activated while 'E' is on, then 'Q' will be changed to on, and if 'D' turns off while 'E' is on, then 'Q' will also turn off.

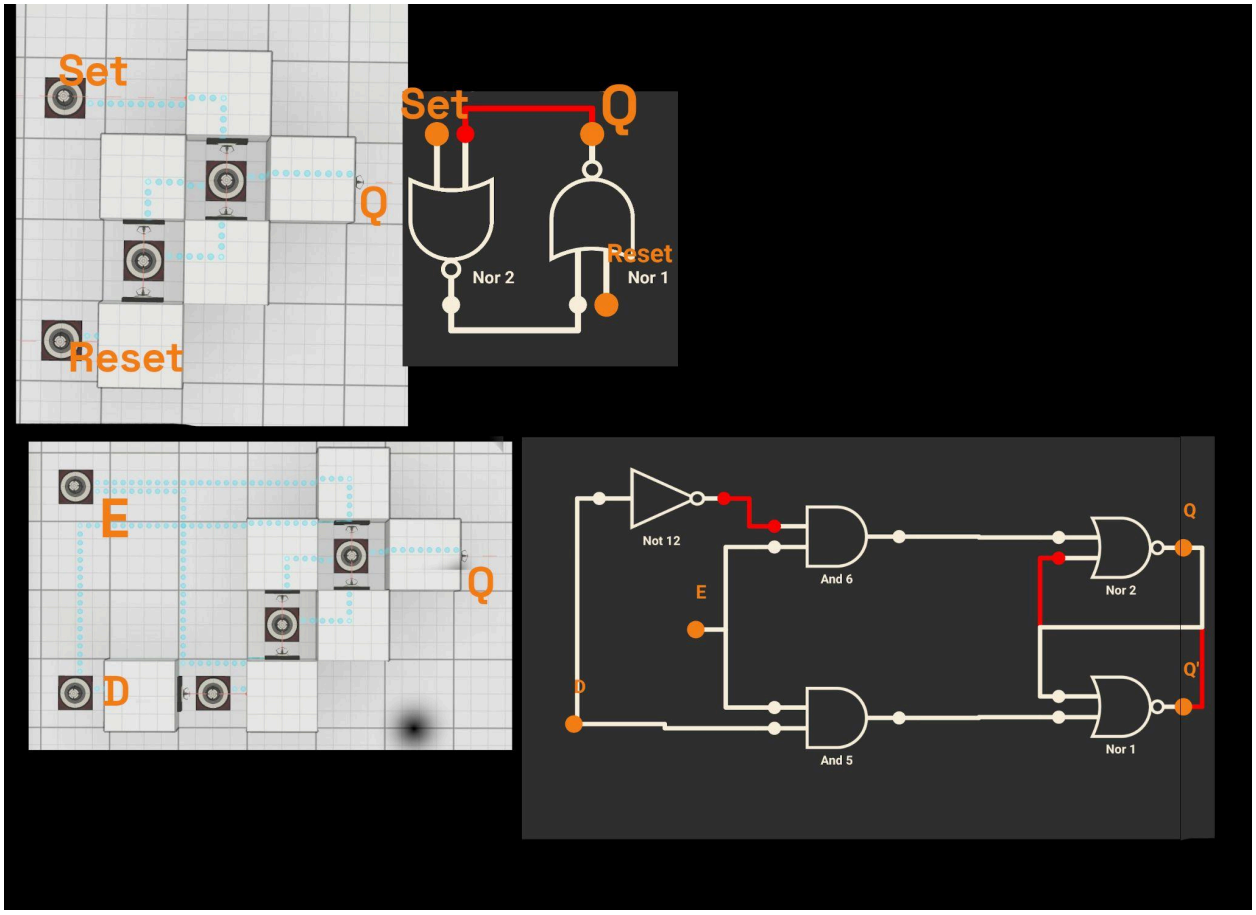



Figure 3.2. The SR latch and D latch implemented in *Portal 2* with their respective circuit diagrams on the right

These circuits can then interact with each other by having their laser outputs point into the input relays of other circuits by following the connections seen in the diagrams. This means that the basic circuits can be used to construct the more complex components of the Turing machine just by following the circuit diagrams. The actual compactness and efficiency of the implemented design depends on the creativity of the puzzle maker, and even the previously mentioned designs can still be improved upon; however, the following designs should theoretically work no matter how they are implemented, as long as there is sufficient space.

The first design seen in figure 3.3 shows a way to simulate a single cell in the Turing machine's tape. This design specifically holds two bits of information which means that it can

store four possible different symbols. This can be expanded upon by simply adding more D latches for more bits, though two bits should be sufficient for simulating a UTM(7,4) as it only uses the symbols 0,1,b, and c which can be represented in the memory cell as the binary values 00,01,10, and 11 respectively (Rogozhin, 225). The SR latch in this circuit tells whether or not this cell in particular is the one currently being read on the tape. Without getting much deeper into the specifics of this circuit, this design allows one to simulate the moving of the Turing machine tape without physically moving the tape and thereby avoiding having to use moving parts in the design. One can possibly build this in *Portal 2* with the input nodes as relays on the tops of all of them and lasers passing through the respective row of relays.

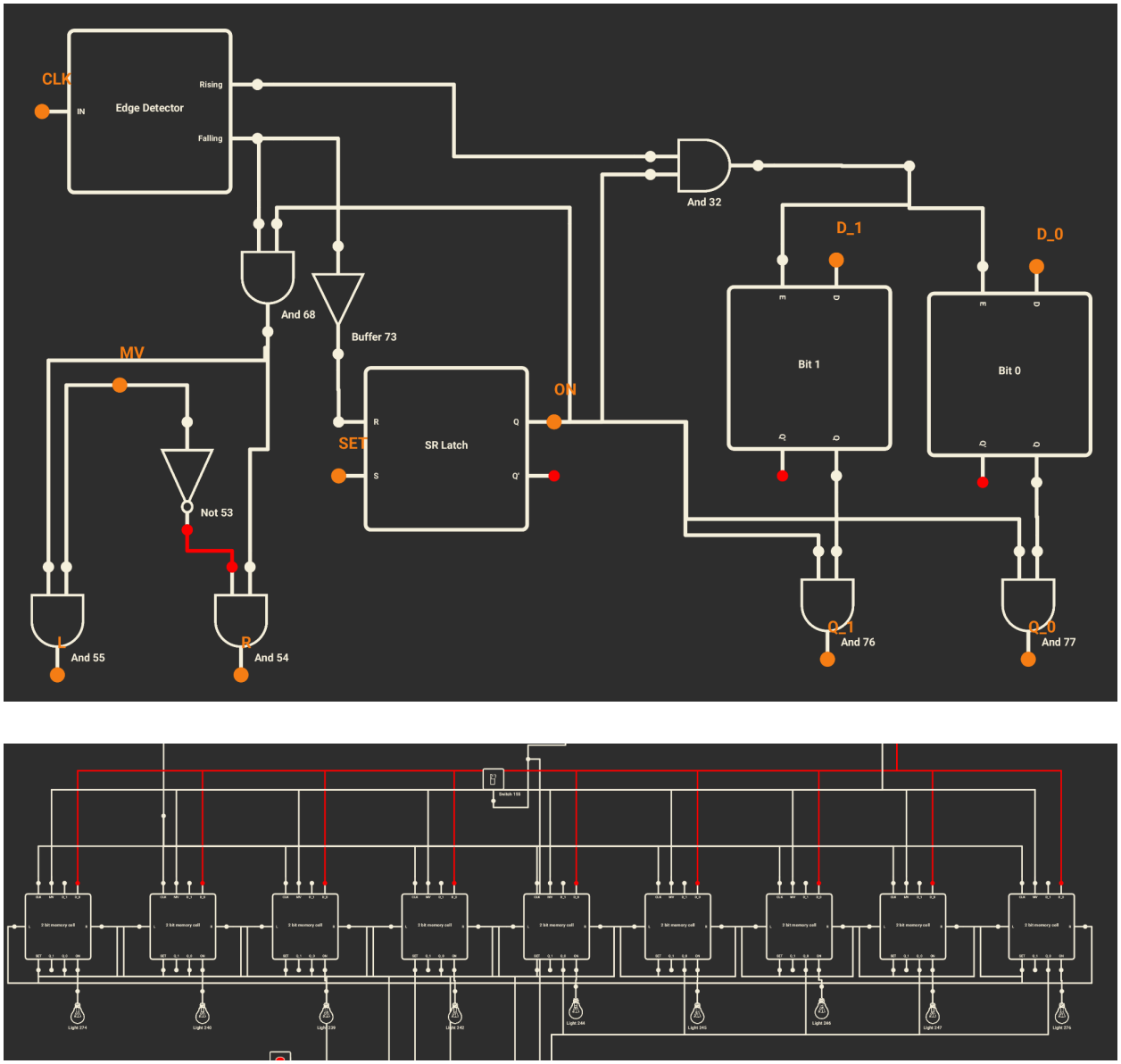



Figure 3.3. The schematic for a single 2-bit memory cell as well as how multiple cells can be connected together to form the Turing machine tape

The symbols are not the only part of the machine that can be represented in binary values. The internal states, or in the case of the factory worker analogy, the pages of the book can also be represented in binary. For example, the 3-state busy beaver from figure 1.2 can have its three internal states each represented by a two bit binary value with A being 00, B being 01, and C being 10 with the value 11 left to represent the halting state which stops the machine. Figure 3.4

shows a truth table representing the rule table for the 3-state busy beaver with the left side showing all the possible combinations of the current state and the symbol currently being read all represented in binary and the right side showing the next state to visit, the symbol to write on the tape, and the direction to go next on the tape with zero representing right and one representing left. Keep in mind that 11 represents the halting state or the state in which the machine stops, so there is no output to the right of 11. Below the table is a logic circuit implemented in *Portal 2* that simulates all the functions described in the truth table as a laser logic array. Unfortunately, the size of this design was already approaching the limits of the puzzle creator so using a similar attempt for the much larger and more complex universal Turing machine would not be possible; however, this does prove that it can be done if the restrictions are somehow lifted. Figure 3.5 shows a similar truth table but for UTM(7,4). The seven states of this universal Turing machine can be represented with three bits or three D latch circuits to store the current state, and since Rogozhin did not include the halting state, that allows the binary value 111 to fit it into the logic which means that the entire input for this machine can be represented in five bits (Rogozhin, 226). Compared to the other UTM designs listed in his paper, UTM(7,4) requires the fewest bits for their logic circuit when taking this approach, which makes it the most ideal design to attempt to build in *Portal 2*.

| Q1 | Q0 | P | s1 | s0 | w | D |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | | | | |
| 1 | 1 | 1 | | | | |

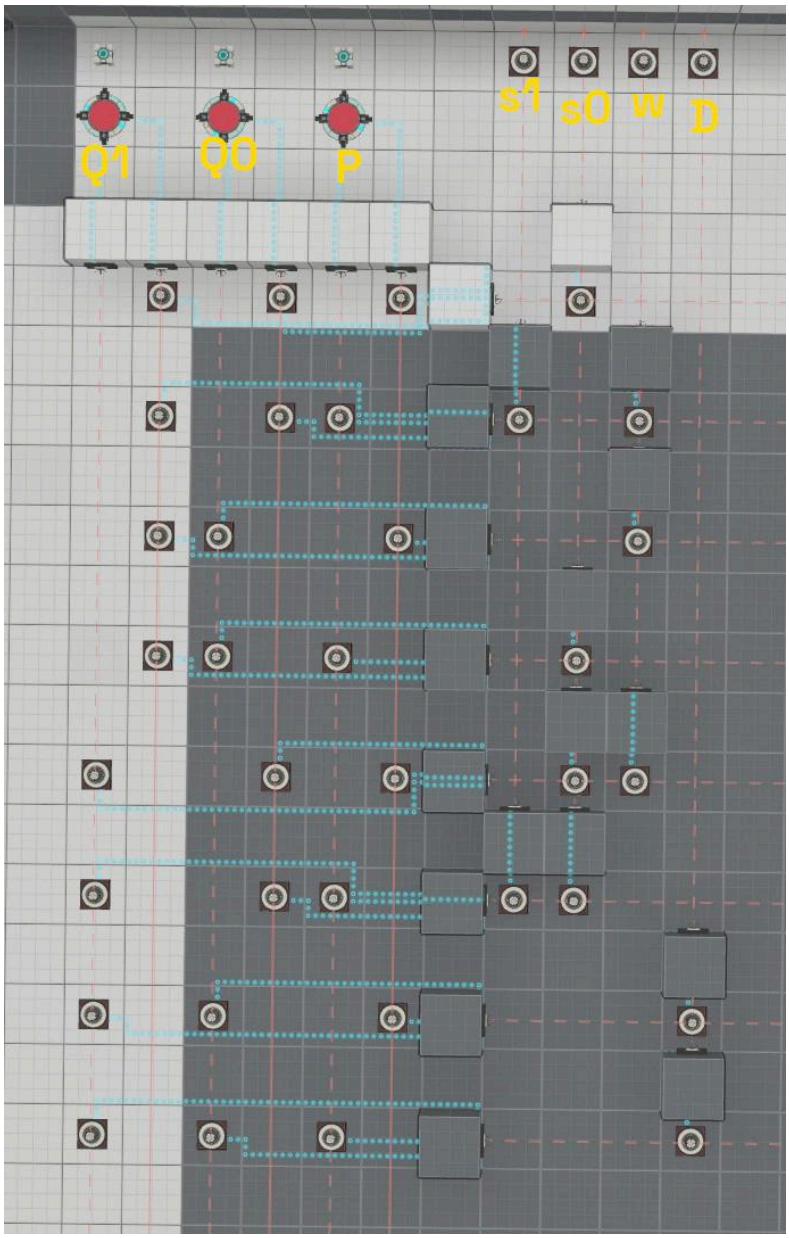



Figure 3.4. The truth table representing the logic for the 3-state busy beaver seen in figure 1.2 as well as an implementation of the logic as a circuit in *Portal 2*

| Q2 | Q1 | Q0 | P1 | P0 | s2 | s1 | s0 | w1 | w0 | D |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | | | | | | |
| 1 | 1 | 1 | 0 | 1 | | | | | | |
| 1 | 1 | 1 | 1 | 0 | | | | | | |
| 1 | 1 | 1 | 1 | 1 | | | | | | |

Figure 3.5. The truth table representing the logic for UTM(7,4)

Due to the complexity of this task, a functioning design for a universal Turing machine could not be developed in time; however, a prototype schematic for a working 3-state busy beaver was designed using the previously mentioned designs. As seen in figure 3.6, the prototype still contains some debugging tools and displays, but when simulated in a program, the design works as intended, and since it is entirely built from circuits that have been shown to be implementable in *Portal 2*, it is theoretically possible to construct this in the game if the size limit is ignored. The component labeled “Logic” in the schematic represents the logic circuit

mentioned in figure 3.4, and since the logic circuit for UTM(7,4) can be constructed in a similar way, it is theoretically possible to use this same layout to also construct the universal Turing machine just by rewiring a few components and adding another D latch at the top to represent the third bit for the seven states of the machine. If one can construct this design in *Portal 2* with the limitations of the in-game puzzle creator, then the game can be confirmed to be Turing complete.

**Conclusion**

It may seem quite redundant at first to construct a computing machine inside of a game that already runs inside a computer. After all, it is much more practical to just use that exact computer to perform calculations several times faster than the simulated machine. Even as a proof of concept, one may see it as a useless gimmick that could never be used for any practical applications. Indeed, there is no actual reason to use *Portal 2* to perform the tasks of a computer; however, the point of this experiment was not to provide a possible alternative to modern computing technologies but to instead provoke thought and discussion on the nature of Turing complete systems themselves. Mainly, it provides a challenge to others looking to learn more about computing systems and who may decide to further expand on the design presented in this paper to construct a more efficient Turing complete system, and the knowledge gained from improving the design can then be used by someone seeking ways to innovate and improve on more practical systems and make real life computers work better than before. Every proof of concept seems like an impractical gimmick at first until someone later down the line is inspired by it and makes something useful, or even continues the chain of inspiration and builds upon the concept for a more practical invention to arise from it. Even the Turing machine itself can be argued to be another example of an impractical concept with no real world value as there is no way to build a machine with an infinitely long tape, yet the concept led to decades of innovations

that eventually led to the creation of the server on which this paper is stored and the computer used to write it.

On top of that, this project also adds to a more philosophical discussion on the infinite nature of computing systems. With a universal Turing machine in *Portal 2*, one can theoretically simulate the game within the machine and build the same machine within the simulated game and repeat the process for an infinite number of iterations if given infinite resources and time. It is this concept of infinite self-replication that allows the existence of things like virtual machines, programs designed to emulate computers within computers that could also theoretically infinitely self-replicate given infinite computing capabilities. These computing systems exist in a physical and finite world, yet they represent the infinite and impossible reality of the Turing machine, and no matter how advanced they get, they could never fully replicate the entirety of the Turing machine, yet we keep pushing forward towards this infinite horizon, this edge of eternity, as we continue to innovate and improve technology to get closer to the concept, but never truly reach it. Zhang Hong takes a similar stance to this perspective in his paper comparing the infinite concept of the Turing machine to Russell's Paradox, another example of an infinity problem in mathematics, stating, "The essential idea is that it is impossible to judge and conclude an evolving infinite world. We can't generate all sets at some point in time, nor can we generate all Turing machines at some point in time" (Hong, 6). The impossibility of these infinities does not stop the quest for knowledge, but rather encourages it. We know it is futile to try and bring to life the infinite world in which the Turing machine resides, yet we still seek it, chasing after answers that may never arrive because the questions are what drive us forward. As such, the quest to prove that *Portal 2* is Turing complete is not one that aims to be concluded with a definitive answer, but rather to bring about more questions and push forward the cycle of knowledge. If the

findings of this paper inspire more questions to be asked and answered, then the true goal of this paper will have been achieved.

**Works Cited**

Bungle. "Portal 2 Logic Gates (NOT, AND, NAND, OR, NOR, XOR, XNOR) (Redstone Computer)." YouTube, 15 May, 2012, https://www.youtube.com/watch?v=d1q72nf8oF0.

Churchill, Alex, Stella Biderman, and Austin Herrick. "Magic: The Gathering is Turing Complete." , 2019, https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=edsarx&AN=edsarx.1904.09828&site=eds-live&scope=site&custid=ken1.

Hong, Zhang. "Turing Machine Halting Problem, Russell's Paradox and the View of Dialectical Infinity." Philosophy of Mathematics Education Journal, no. 39, 2022, pp. 1-6, https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=eue&AN=164327029&site=eds-live&scope=site&custid=ken1.

Hull, Thomas C., and Inna Zakharevich. "Flat origami is Turing Complete." , 2023, https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=edsarx&AN=edsarx.2309.07932&site=eds-live&scope=site&custid=ken1.

Kaye, Richard. "Infinite versions of minesweeper are Turing-complete." Manuscript, August, 2000.

Leavitt, David. The Man Who Knew Too Much. W. W. Norton & Company, 2006.

MystX1. "Portal 2: 4-bit adder and 7-segment hex display." YouTube, 12 July, 2012, https://www.youtube.com/watch?v=AWZBmmVjPgo.

Rogozhin, Yurii. "Small universal Turing machines." Theoretical Computer Science, vol. 168, no. 2, 1996, pp. 215-240, https://www.sciencedirect.com/science/article/pii/S0304397596000771, doi:10.1016/S0304-3975(96)00077-1.

"Turing Machine Visualization." , https://turingmachine.io/.

Turing, A. M. "On Computable Numbers, with an Application to the Entscheidungsproblem." Proceedings of the London Mathematical Society, vol. s2-42, no. 1, 1937, pp. 230-265, https://doi.org/10.1112/plms/s2-42.1.230, doi:10.1112/plms/s2-42.1.230.

Valve Software. "Portal 2." , 2011.

Yaga, Dylan, et al. "Blockchain Technology Overview.", 2019, https://explore.openaire.eu/search/publication?articleId=od_____18::5775a0faab406f22872774ad8b082646, doi:10.6028/NIST.IR.8202.