Human Intelligence vs. Machine Logic

New Mexico Supercomputing Challenge
Final Report
April 23rd, 2019
Multi-School 12
Nex+Gen Academy, Eldorado High School

Team Members:
Brendan Kuncel, Quentin Dye, and Savannah Phelps
Sponsor Teacher:
Karen Glennon
Mentor:
Patty Meyers

Table of Contents

Executive Summary

It is common knowledge that a brain is comparable to a computer due to basic similarities in design. However, these principles have critical differences that lead to the disparity between the skills computers and brains can have. For example, a child can read and comprehend the meaning of a simple sentence using far fewer neural steps than lines of code a computer would have to use to perform the same task. However, a computer can perform complex equations in milliseconds that would take a human minutes or hours to comprehend and correctly solve.

Unfortunately, although we can understand these design differences, we can not as easily implement them. As an example, human brains are superior at parallel processing, while computers are superior at serial processing. Parallel processing will always trump serial when enough steps are required, but parallel processing is extremely difficult to implement hardware-wise, which is why it is only used in supercomputers, and even then relatively limited.

Studying the complexities between humans and machines gives rise to many questions. Such as, in our case, what will these differences change when humans solve the same maze as machines.

Machines are programmable, and perfect in the respect that they follow their commands exactly. We know for a fact that artificial intelligences will not include bias and will strictly follow an algorithm. This is something we can not know about humans, who are prone to suspicion.

For example, a human may pick directions to go in a maze at random, only attempting to head in the direction of the end. They may suspect a certain direction will lead them closer, and pursue that direction until barriers dissuade them.

Our hope is that enough of human’s maze solving techniques are relatively systemized, or at least show resemblances to other’s techniques so that we can make a model of a human based off of this data. This model in itself has fallacies, including the design differences mentioned above being breached.

Our project therefore, is to examine these differences and try to overcome these difficulties.

Introduction

Problem Definition

Our project is to create a comparison of human intelligence and machine intelligence in the scopes of iterative maze solving. We will analyze how a human and an agent-based model improve the time it takes for them to solve in terms of the number of turns they take.

This will involve creating a maze solver in netlogo that can incorporate multiple maze solving algorithms, a program that tracks the number of steps, finds the optimal maze path and performs other record keeping functions, and a clear user interface that allows the human to solve the maze on a computer.

We will also test human subjects with varying features such as handedness, age, personality, gender, race, and background in order to remove as much situational bias from the human testing.

Background

Last year, our team completed a project on the efficiency of different maze-solving algorithms: flood method, right-hand, left-hand, clockwise, and counter-clockwise in randomly generated and grid mazes. We discovered that the flood method was most effective, but not very realistic for humans attempting to solve a maze.

This project allowed us to learn about the maze solving techniques we would most like to compare to humans. The algorithms we have chosen to use are all possible to humans, although humans would not follow them.

Research in this area has been done before, but in our searching we have been unable to find a more general comparison such as we are attempting.

Significance

Although this project does not have possible real-world applications in mind, there are possibilities for its use.

This project, for example, demonstrated a long debated point: that humans and machines are far from reaching the same level of maze solving efficiency. To expand, humans and machines can not yet perform the same activities in the same way. Perhaps in the future the limiting factors in our project will be removed, but presently, we are stuck.

Research

Algorithms

Flood Solve: (This is an algorithm separate from what we are testing, we are using this to find the optimal path through the mazes we have created.) This method simply tests every single path through the maze and compares the number of steps each path takes. Although this can get heavy in larger mazes, our mazes are small enough that it is not difficult for the code.

Wall Follower (Right or Left): This algorithm simply always takes a specific direction, which in our program can be changed to right or left, then changes which direction they go counterclockwise or clockwise respectively. This algorithm can get stuck in areas where a loop path occurs, but it mostly reliable.

Tremaux: Also called the breadcrumb method, this algorithm tracks the path the agent is taking through the maze. To begin, it follows the principles of a wall follower algorithm, but when the previously taken path intercepts with where the agent is going, the agent turns around and retraces its steps until a new, untravelled path is found and followed.

This algorithm is unique because while it may not be able to find the optimized path, it can always find a solution regardless of loops in the maze structure.

Problem Solution

Method

In order to solve our problem, we realized that our project would be a comparison of three parts. Part One would be the computer maze solver created with its algorithms listed above. Part Two would be the average human model created by factoring in all the tests that we conducted. Part Three would be a sort of control, and simply contain the optimal path through the maze. We also needed to include code for recording the number of steps a solution took, displaying the maze, and tracking a user or computer's path through the maze.

All of this was done in NetLogo due to its unique ability to visually represent the code we write, and the user's ability to interact with an interface. We have used NetLogo in many of our past projects, including last year's.

Since we have two mazes, we would perform this comparison twice with the different mazes.

Part One involved programming out each maze algorithm we were going to use. As we did this, research from last year came in to play, demonstrating where the three computer algorithms we are using lack compared to humans.

Part Two meant that we needed several changes to how the maze display worked. We had to incorporate directional buttons, physically prevent the user from crossing maze walls, and limit the user's vision throughout the maze. If a user were to try to cross a wall by pressing a button, the command will not work and an error message will appear in a text box beside the maze.

We chose to limit the user's vision as they go through the maze so that they can only see the walls directly around their cursor and the ending square to even the playing field between humans and machines. Most algorithms can also only see the squares around them as well as the beginning and end.

Unfortunately, after we completed human testing, we were unable to digitally analyze results, but through observing the completed mazes and their trends, we were able to objectively map out where many humans would chose to go.

Part Three was very similar to the functions of Part One, but is a separate category of comparison. Finding the optimal path through the maze and comparing the number of steps that path takes to how many Part One and Part Two takes.

Program

To begin, a simple interface is displayed (see 7.1). In this image, one of the previously created mazes is seen. This was created by a sequence of code triggered by setting the variable writing = true. This code allows us to use arrow keys or the buttons on the interface to delete wall segments in a blank grid. These mazes can be stored within the program and remembered later. The following code is a segment from this sequence:

```
if writing = true
  [
    file-close
    file-open Stored_Maze_Name
    file-write "w"
  ]
  ask turtles
  [
    ifelse finished != true
    [
      ask patch (xcor) (ycor)
      [
        set closed-top false compile
      ]
      set heading 0 forward 1
      ask patch (xcor) (ycor)
      [
        set closed-bottom false compile
      ]
    ]
    [
      set heading 0
      ifelse closed-top != true
      [
        ask patch (xcor) (ycor)
        [
          sprout 1
          [
            set shape "up arrow" set heading 0 stamp die
          ]

        ]
        set heading 0 forward 1
        set step step + 1
        ask patch (xcor) (ycor)
        [
          sprout 1
          [
            set shape "down arrow" set heading 0 stamp die
          ]

        ]
        Output-print "\n \n \n \n"
      ]
      [
        Output-print "Action not allowed!"
      ]
    ]
  ]
```

The keystrokes used to create the maze can be stored in NetLogo’s built in file-storage system. They can be ead back by the print-maze function in the interface. Some pitfalls while building this was making sure that only keys while building the maze would be saved. This was solved by adding a variable defining when the maze is being written (writing = true). The file also checks to make sure there are still letters left to read back.

The initial starting and ending points (shown by the green and red squares in the maze) can be selected with numerical inputs for x and y coordinates in the interface.

The following code is a simple command to reveal the walls around it when a human is solving the maze. It asks any patch that has a turtle on it to reveal it’s walls with the function show-walls.

```
To show-walls
if closed-top = true
  [
    sprout 1
    [
      set shape "Top-Wall" set heading 0 stamp die]

  ]
End
```

It is important to note that what is seen in the interface is purely visual. The working code uses variables, not walls, then converts the variables into specially designed patches.

The flood method used to find the optimal path (see 7.2) is done by having turtles check every direction for both walls and other turtles. If none are found in a certain direction, the code will send an identical turtle to that location. This is repeated until there is a turtle on the end patch. When this happens, the function optimize-path is called, showing the final path as the one with the shortest number of steps. As an example, the code for checking the right before extending to that point is as follows:

```
if closed-right != true
      [
        if not any? turtles-on patch-at-heading-and-distance 90 1
        [
          hatch 1 [set heading 90 forward 1]
        ]
      ]
```

Tremaux method (see 7.4) detects a dead end to block off using a set of instructions that give each wall around it a temporary variable. If there are three walls surrounding the turtle, that square is declared a dead end and is blocked off.

The code for this process is as follows:

```
set tempcounter 0
    ifelse closed-top = true
    [
      set tempcounter tempcounter + 1
    ]
    [
      set direction-to-close "top"
    ]
    ifelse closed-bottom = true
    [
      set tempcounter tempcounter + 1
    ]
    [
      set direction-to-close "bottom"
    ]
    ifelse closed-right = true
    [
      set tempcounter tempcounter + 1
    ]
    [
      set direction-to-close "right"
    ]
    ifelse closed-left = true
    [
      set tempcounter tempcounter + 1
    ]
    [
      set direction-to-close "left"
    ]
    if tempcounter = 3
    [
      if direction-to-close = "top"
      [
        ask patch xcor ycor
        [
          set closed-top true
          show-walls
          set gone-up false
        ]
        right-up
        ask patch xcor ycor
        [
          set closed-bottom true
          show-walls
          set gone-down false
        ]
      ]
      if direction-to-close = "bottom"
      [
        ask patch xcor ycor
        [
          set closed-bottom true
          show-walls
          set gone-down false
        ]
        right-down
        ask patch xcor ycor
        [
          set closed-top true
          show-walls
          set gone-up false
        ]
      ]
      if direction-to-close = "left"
      [
        ask patch xcor ycor
        [
          set closed-left true
          show-walls
          set gone-left false
        ]
        right-left
        ask patch xcor ycor
        [
          set closed-right true
          show-walls
          set gone-right false
        ]
      ]
      if direction-to-close = "right"
      [
        ask patch xcor ycor
        [
          set closed-right true
          show-walls
          set gone-right false
        ]
        right-right
        ask patch xcor ycor
        [
          set closed-left true
          show-walls
          set gone-left false
        ]
      ]
      set temp true
    ]
```

```
Now that the algorithms are created, one last function is needed to run them,
which is as follows:

to run-solver
  if solver = "Left-Hand"
  [
    left-hand
  ]
  if solver = "Right-Hand"
  [
    right-hand
  ]
  if solver = "Flood"
  [
    Flood
  ]
  if solver = "Tremaux"
  [
    breadcrumb
  ]
  if finished? = true
  [
    stop
  ]
end
```

Human Testing

Human testing was perhaps the most difficult part of this project to approach. Humans have too many metal processes running at all times to create a perfect testing environment. The only way to try to remove this bias is to collect a wide variety of data, and let the differences begin to neutralize each other.

Some examples include what instructions are given to each test subject (telling them to focus on speed or accuracy can completely change their solving technique), how familiar they are with the testing software, and any number of outside stressors.

At the beginning, we were concerned with how to provide the computer and the human with as similar a situation as possible, which led to us removing the humans ability to see the entire maze. Any further handicaps threatened to create more uncontrollable variables.

Our testing procedure is as follows:

1. Place subject in a comfortable seated position in front of the computer displaying a blank test screen (see Appendix).
2. Instruct subjects: “A maze will appear on the screen in front of you. Use W,A,S, and D to move up, left, down, and right respectively. You may not go through walls. If you try to, an error will appear. You will be able to see the beginning and the end, the walls directly around you, and where you have previously been. You are being measured on accuracy, not speed. Please try your best.”
3. Administer the two mazes sequentially, photographing and labeling their completed maze after each trial.

Challenges

Our project has been through many iterations and changes throughout the year. Early in the project, our project involved creating a robot that would physically solve mazes and improve its effectiveness. This idea was scrapped in favor of something with a more quantitative “solution”.

Later, our project required us to create code that goes beyond our physical means and understanding of artificial intelligence. We intended to use principles of reinforcement learning, something tricky to comprehend, let alone program and apply to our overall project goal.

We also had become sidetracked trying to make our own maze generator. We realized from mentor critique that this was an unnecessary feature we could easily find other software for or manually create our mazes (what we eventually did).

Many small decisions or roadblocks appeared, but nothing that didn’t strengthen the validity of our project.

Conclusions

Results

At the conclusion of this project, after having testing our model using computer algorithms and humans, we have discovered the following facts:

- The Wall Follower methods are impractical in this setting, and were never able to solve the mazes we created due to there being a loop
- While the Tremaux's method is more reliable than humans in terms of maintaining a consistent number of steps throughout tests, humans have been much closer to the optimal path more times
- Humans step counts show the highest variety between test subjects and mazes
- Humans favor vertical directions before horizontal, and regardless of the locations of the starting and ending point will usually go up or down towards the finish

Analysis

In the grand debate between humans and machines, it appears that in this limited scenario, humans advanced parallel processing and creative thought trump the computer's steadiness.

In a way, our findings support what scientists and engineers have been realizing since 1965 when the first artificial intelligences capable of solving calculus problems and playing chess were created. At this time, many believed that in a few decades robots would catch up to humans (which we now know was quite the understatement).

Future Development

This project could be benefitted extremely by widening the scopes used to test with. More widespread testing combined with more computer algorithms would make this a more expansive and accurate conclusion.

In future years, we would follow these suggestions as well as possibly incorporate more of our research regarding human learning.

Acknowledgements

We would like to thank Patty Myers and Karen Glennon for always supporting our efforts, giving us advice and keeping us on track. We also appreciate our interim judges, Zeinab Akhavan, Scott Wilson, and Quincy Wofford for helping us get a better perspective on our project and look at ways to improve.

Team Information

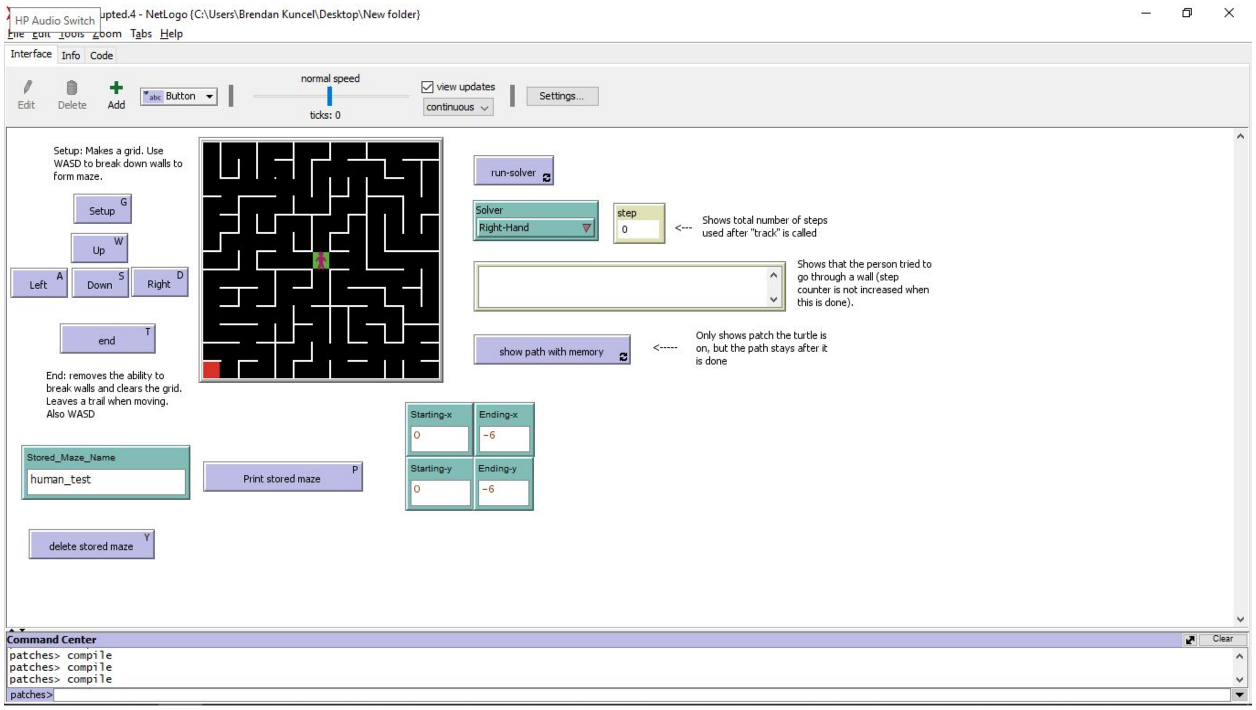
Brenden Kuncel is a junior attending Eldorado High School. He is very interested in STEM fields, having been in the challenge or GUTS since elementary school. He was the main programmer in this project.

Quentin Dye is a senior at NexGen Academy. He is planning on attending New Mexico Tech next year, and currently works for the challenge designing this year’s website. He is the secondary programmer, researcher, and website designer.

Savannah Phelps is a sophomore at Eldorado High School. She is interested in mathematics and robotics at her high school, as well as how these topics apply to fine arts. She is the report writer and main researcher.

Appendix

1. Maze interface with one of our premade mazes



HP Audio Switch upted.4 - NetLogo {C:\Users\Brendan Kuncel\Desktop\New folder}

Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Stored_Maze_Name: human_test

Solver: Right-Hand

step: 0

<--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

Starting-x: 0

Ending-x: -6

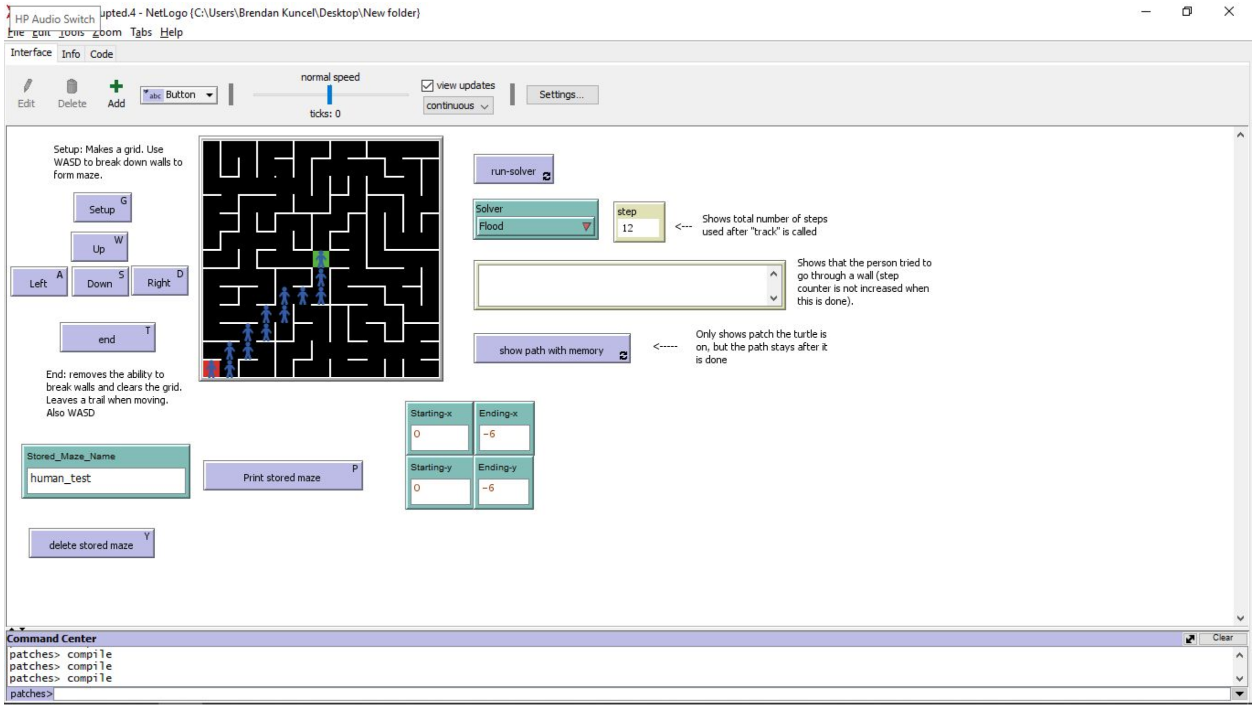
Starting-y: 0

Ending-y: -6

Command Center

```
patches> compile
patches> compile
patches> compile
patches>
```

2. The optimal path for both of the mazes found by using the flood solve method



upted.4 - NetLogo {C:\Users\Brendan Kuncel\Desktop\New folder}

Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Stored_Maze_Name: human_test

Solver: Flood

step: 12

<--- Shows total number of steps used after "track" is called

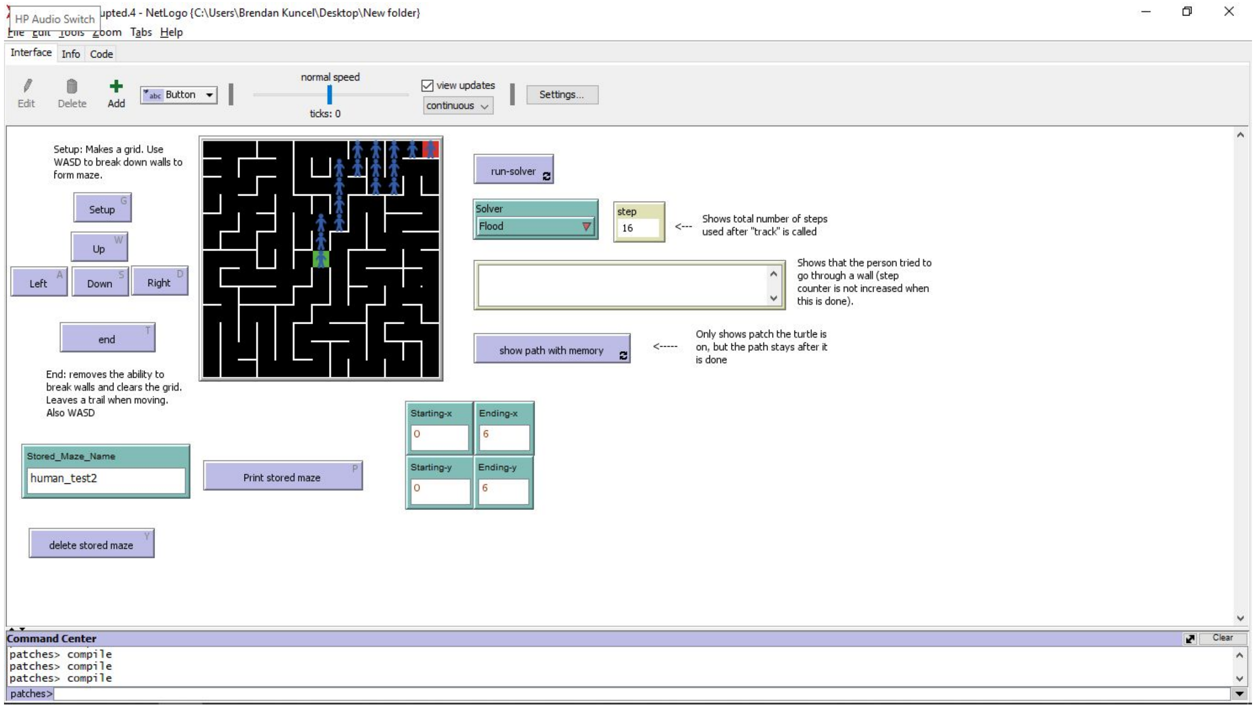
Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

Starting-x: 0 Ending-x: -6

Starting-y: 0 Ending-y: -6

```
patches> compile
patches> compile
patches> compile
```



upted.4 - NetLogo {C:\Users\Brendan Kuncel\Desktop\New folder}

Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Stored_Maze_Name: human_test2

Solver: Flood

step: 16

<--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

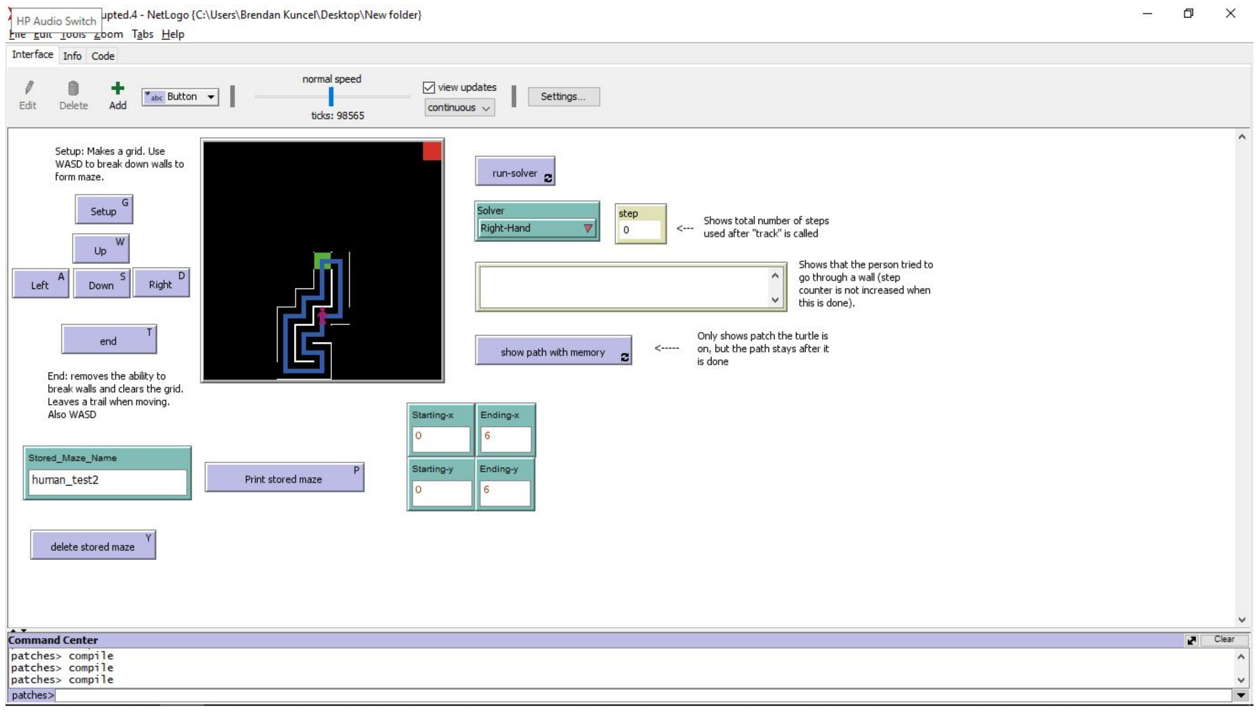
<----- Only shows patch the turtle is on, but the path stays after it is done

Starting-x: 0 Ending-x: 6

Starting-y: 0 Ending-y: 6

```
patches> compile
patches> compile
patches> compile
```

3. The path used by the left and right wall follower methods on one of our mazes



normal speed
ticks: 98565
view updates: ✓ continuous

Setup: Makes a grid. Use WASD to break down walls to form maze.

Solver: Right-Hand

step: 0 <--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

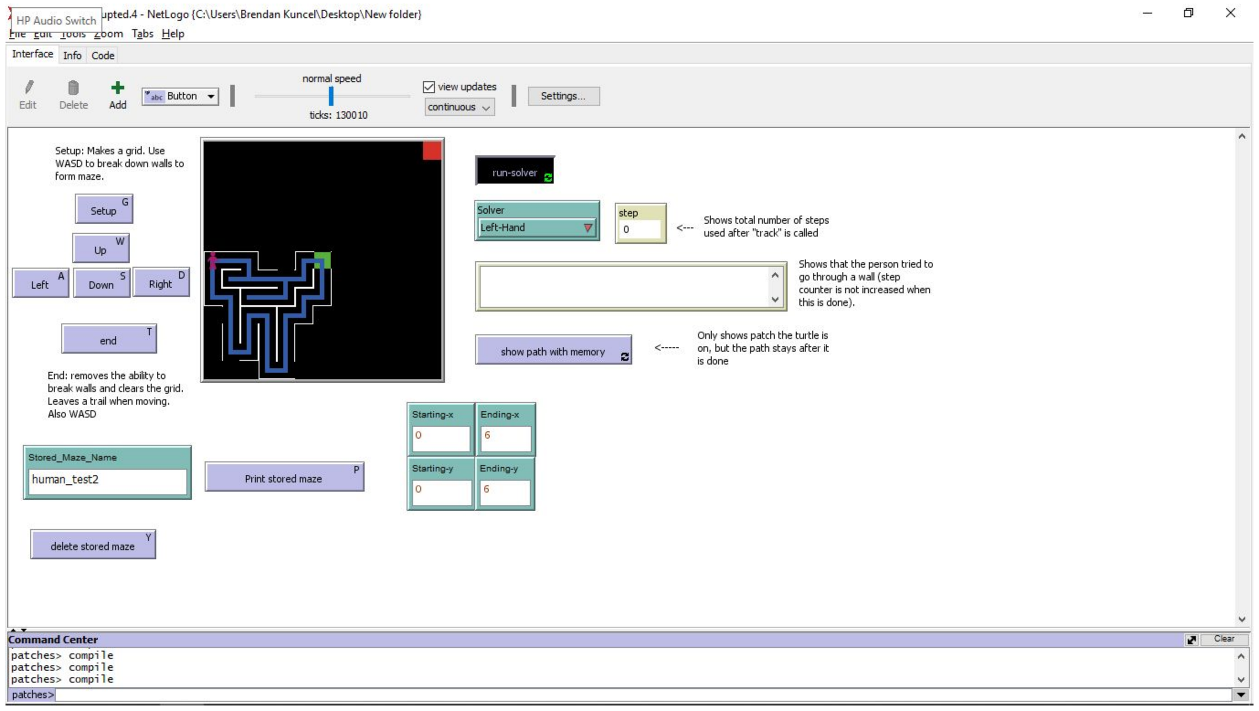
<----- Only shows patch the turtle is on, but the path stays after it is done

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Starting-x: 0
Ending-x: 6
Starting-y: 0
Ending-y: 6

Stored_Maze_Name: human_test2

```
patches> compile
patches> compile
patches> compile
```



normal speed
ticks: 130010
view updates: ✓ continuous

Setup: Makes a grid. Use WASD to break down walls to form maze.

Solver: Left-Hand

step: 0 <--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

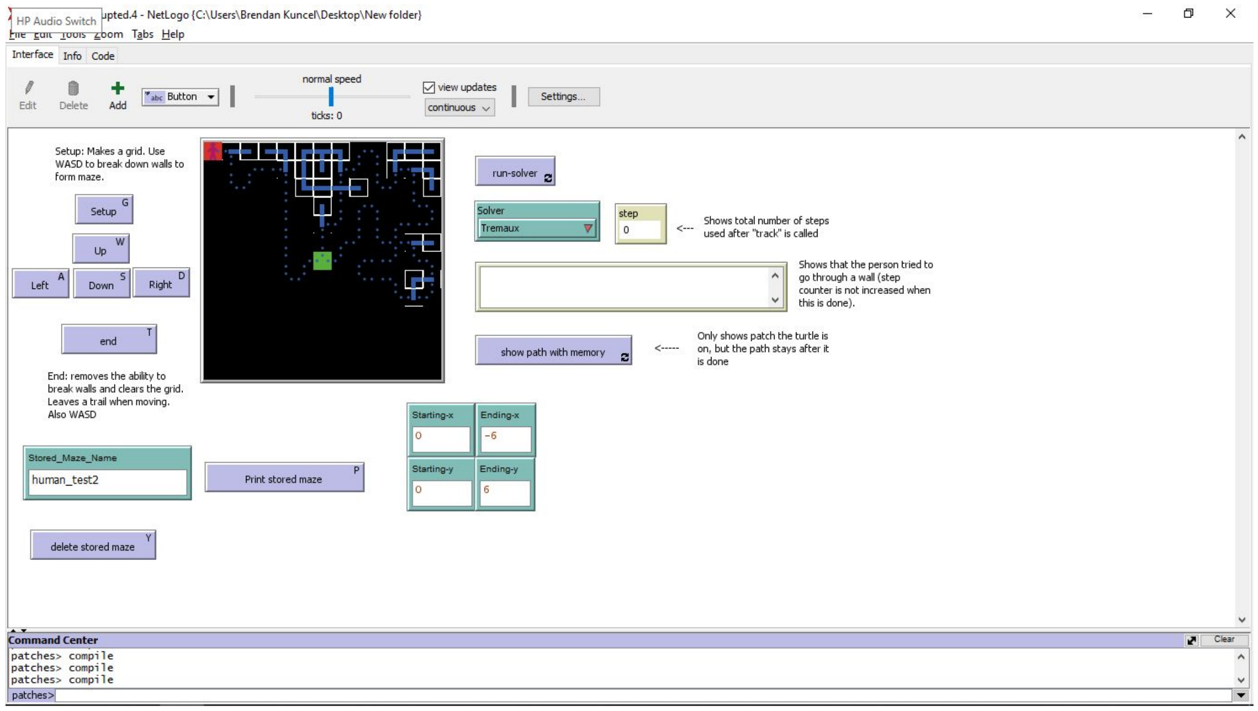
End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Starting-x: 0
Ending-x: 6
Starting-y: 0
Ending-y: 6

Stored_Maze_Name: human_test2

```
patches> compile
patches> compile
patches> compile
```

4. The path found by Tremaux's method in one of our mazes



normal speed

ticks: 0

☑ view updates

continuous

Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Solver: Tremaux

step: 0 <--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

Stored_Maze_Name: human_test2

Starting-x: 0

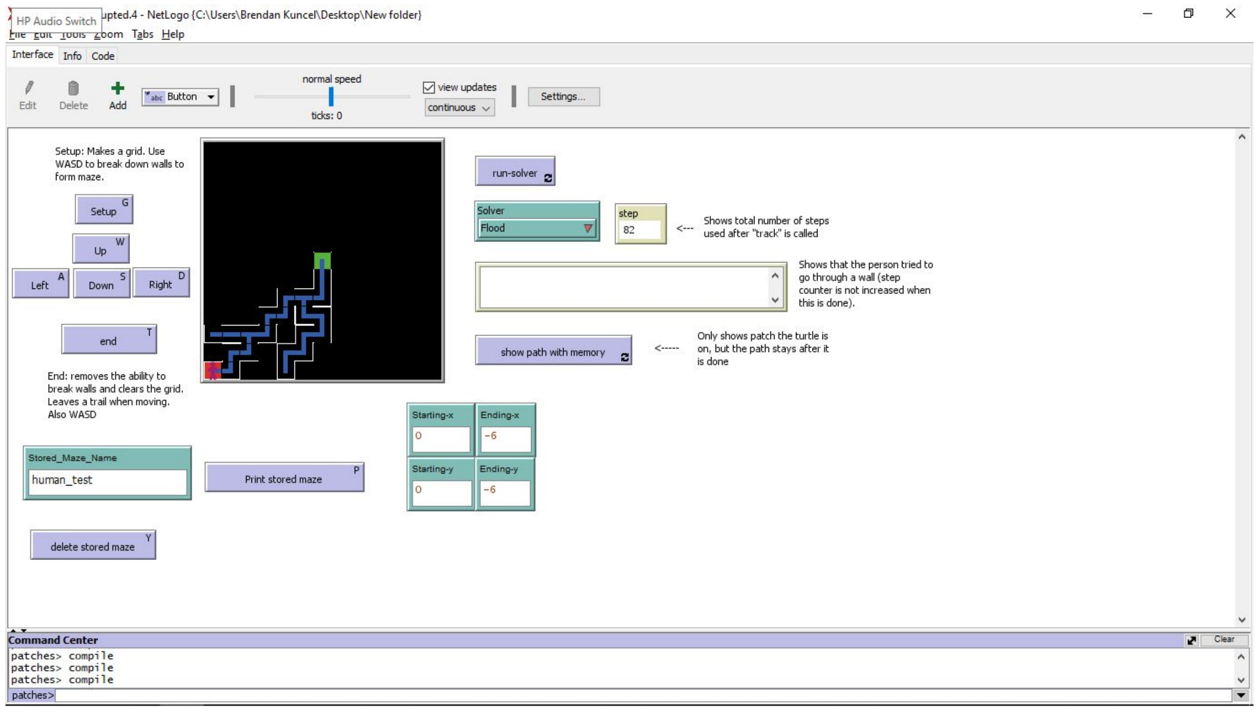
Ending-x: -6

Starting-y: 0

Ending-y: 6

```
patches> compile
patches> compile
patches> compile
patches>
```

5. Both mazes solved by a sample human test subject



normal speed

ticks: 0

☑ view updates

continuous

Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Solver: Flood

step: 82 <--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

Stored_Maze_Name: human_test

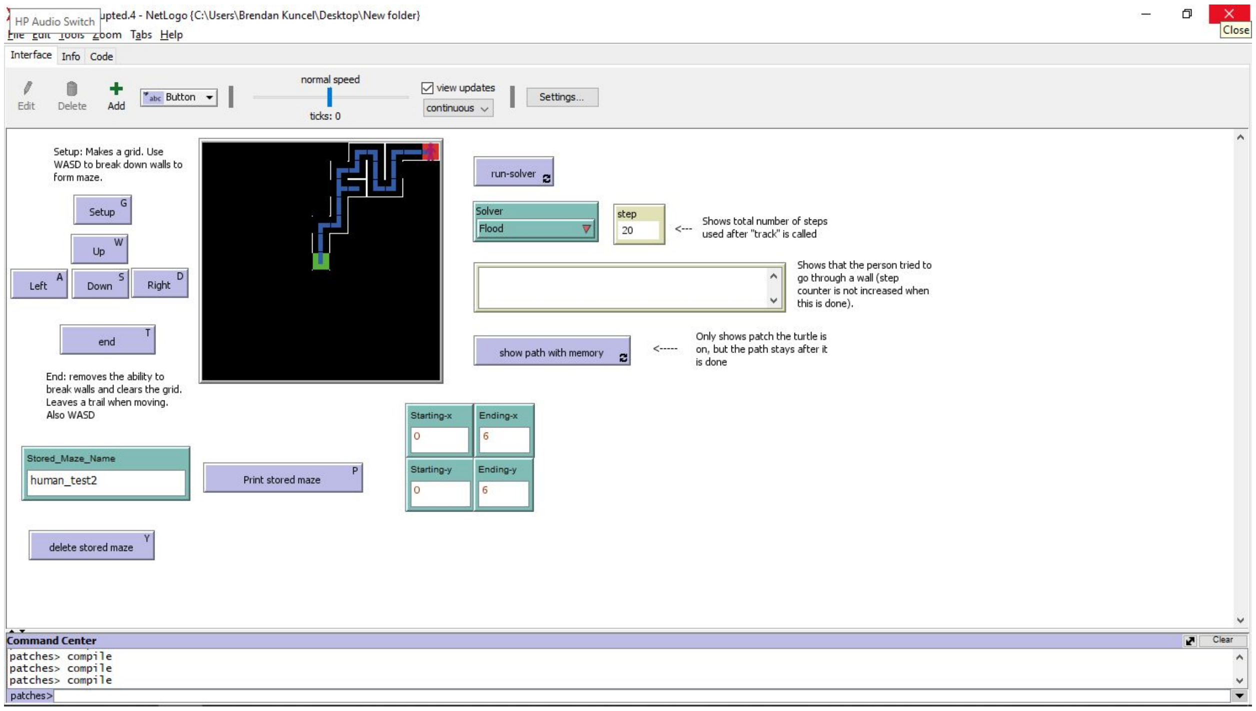
Starting-x: 0

Ending-x: -6

Starting-y: 0

Ending-y: -6

```
patches> compile
patches> compile
patches> compile
patches>
```



Setup: Makes a grid. Use WASD to break down walls to form maze.

End: removes the ability to break walls and clears the grid. Leaves a trail when moving. Also WASD

Stored_Maze_Name: human_test2

Solver: Flood

step: 20

<--- Shows total number of steps used after "track" is called

Shows that the person tried to go through a wall (step counter is not increased when this is done).

<----- Only shows patch the turtle is on, but the path stays after it is done

Starting-x: 0

Ending-x: 6

Starting-y: 0

Ending-y: 6

```
patches> compile
patches> compile
patches> compile
```

Bibliography

1. Nilsson, Nils J. *Principles of Artificial Intelligence*. Springer Berlin, 2014.
2. Moravec, Hans P. *Mind Children: the Future of Robot and Human Intelligence*. Harvard Univ. Pr., 2010.
3. Minsky, Marvin. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon & Schuster, 2007.
4. Washburn, David A. “Analyzing the Path of Responding in Maze-Solving and Other Tasks.” *Behavior Research Methods, Instruments, & Computers*, vol. 24, no. 2, 1992, pp. 248–252., doi:10.3758/bf03203502.
5. Whitworth, Brian, and Hokyoung Ryu. “A Comparison of Human and Computer Information Processing.” *Machine Learning*, 2012, pp. 1–12., doi:10.4018/978-1-60960-818-7.ch101.