Learning Recursion with 3D Turtle Graphics in a Virtual Reality Environment

by

Joshua Sun

Dr. Davide Fossati
Adviser

Department of Computer Science

Dr. Davide Fossati

Adviser

Dr. Michelangelo Grigni
Committee Member

Dr. Elizabeth Newman

Committee Member

2025

Learning Recursion with 3D Turtle Graphics in a Virtual Reality Environment

By

Joshua Sun

Dr. Davide Fossati
Adviser

Department of Computer Science

2025

Abstract

Learning Recursion with 3D Turtle Graphics in a Virtual Reality Environment
By Joshua Sun

Turtle graphics is a pedagogical tool that has been proven effective to teach various programming concepts. In its original formulation, students write programs that move an automaton (the Turtle) on a 2-dimensional (2D) screen using sequences of commands. There have been various attempts at creating 3-dimensional (3D) versions of the turtle; and, to our knowledge, there is currently only one 3D programming turtle in Virtual Reality (VR). Recursion is a fundamental and challenging concept in Computer Science education. While the traditional 2D Turtle has been widely used to teach recursion, using a 3D Turtle is way less common for this purpose.

We designed a sequence of activities that students would do to learn recursion, using the 2D Turtle. Next, we then developed alternatives that work in 3D and VR. After, we then developed a 3D Turtle that could be viewed in VR to fit all the activities. We tested our system with two groups of students: 2D, and 3D/VR groups. Students completed a pre-survey and pre-quiz, performed the activities, and took a post-quiz and post-survey.

From our study, we saw that, while students did improve in their understanding of recursion, there was very little difference in improvement between the 2D and 3D/VR students. Our experiments highlighted several lessons in using the 3D/VR Turtle. First, compared to 2D, 3D rotations are much more difficult to visualize, and they require students to be comfortable rotating axes and bases. Second, the usage of VR should be more streamlined: students were asked to code in a text editor, and take the VR headset on and off while working through the activities. The barrier between coding and viewing the turtle reduced enjoyment of the activity. Third, our activities may have not used the 3D-VR space to its fullest potential. This gave us more ideas on future activities tailored more specifically for the 3D/VR Turtle.

Learning Recursion with 3D Turtle Graphics in a Virtual Reality Environment

By

Joshua Sun

Dr. Davide Fossati
Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Computer Science

2025

Acknowledgments

Thank you to Dr. Fossati for assisting in, what is essentially, a game development project. Thank you to the Godot Foundation for making a game engine free and open-source. Thank you to friends and family that have helped me get through my panic of finishing this.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Turtle Graphics, or just "the turtle" for short, is a common teaching tool for introductory programming. It usually involves a "turtle," a small graphic representation of some object, moving around to draw lines. When the turtle moves, it draws a line behind itself. The turtle can move forward and backward, rotate clockwise and counterclockwise, color the line, and toggle the line on and off. A programmer can then issue those commands to the turtle using a programming language. By visualizing certain actions a turtle takes, students are able to see what their code is doing, as opposed to abstractly just receiving some data. Students are able to understand unfamiliar programming concepts by working with a tangible, visible object.

Virtual Reality (VR) as a technology allows a user to visualize virtual three-dimensional(3D) spaces more clearly. By using the headset, they become more immersed within the constructed world. The immersion of VR combined with the concepts of visualization and construction of the turtle makes for an obvious pair. However, there seems to be no research on the benefits or effectiveness of using a VR turtle. The main force behind this paper is the pedagogy of visualizing the turtle in 3D and VR, and determining whether if there is a benefit to learning.

To investigate the benefits of VR, we narrowed down our focus to only one computer science concept. One of the more difficult concepts is recursion. Students typically struggle with understanding recursion, as it is much more of an abstract concept, compared to other concepts such as variables or iteration. We chose recursion because it is a core computer science concept taught in many introductory courses. It is fundamental to many other computer science topics, such as data structures (i.e. linked lists, graph traversal). Recursion is also in the sweet spot between being "too easy" to "too hard" to notice any tangible results.

This leads us to the question of this thesis: How does teaching with a 3D/VR Turtle benefit or hinder a student's ability to understand recursion compared to the two-dimensional (2D) Turtle?

We studied the impact of VR through this preliminary study. First, we developed a series of activities, a turtle program viewable in VR, quizzes, and surveys for students to carry out. Experimentally, we gathered students to complete the quizzes, surveys, and the series of activities designed to be completed with the turtle. As this study is designed for education at the undergraduate level, the students were undergraduate students. After, a qualitative analysis was done, looking at student behavior, comparing quizzes, and general feedback from the surveys. We then discuss the limitations of the study, as well as various adjustments and improvements that could be made.

Analyzing the data, 3D/VR did not harm understanding, but it did not seem to benefit more compared to the 2D version. Students were able to answer more confidently and correctly on the post-quiz than the pre-quiz. However, the results were similar between the 3D/VR versions. If one were given the pre-quiz and post-quiz, they would not be able to determine whether the student completed the 2D activities or the 3D/VR activities. Overall, we concluded that, while 3D/VR turtle did not reduce a student's ability to understand recursion, it did not improve compared to the 2D turtle.

Chapter 2

Background

2.1 Difficulties in Teaching on Recursion

Recursion is a fundamental concept usually taught early on in computer science education. It is known to many educators that teaching recursion is a difficult task. At the same time, for many beginners, it is difficult to understand recursion as a whole. There is plenty of literature that focuses on various methods of teaching recursion.

Hazzan, Ragonis, and Lapidot issue an activity based approach. It concerns itself with direct activities and tasks assigned to learners. For example, one of the activities was classification and research. They asked students to look out into the world and discuss whether things they saw were recursive or not. Another approach was to discuss various models to visualize recursion. One model was the top-down frames model, seen in figure 2.1. It visualizes the path of computation the recursive function took by describing each recursive call of the function as a separate computational frame. That frame was then placed on top of the previous frame continuing as such. They point out that many learners have difficulty separating the recursive algorithm with the recursive process. [5]

Gunion, Mildford, and Stege attemped to teach recursion at a younger age. They

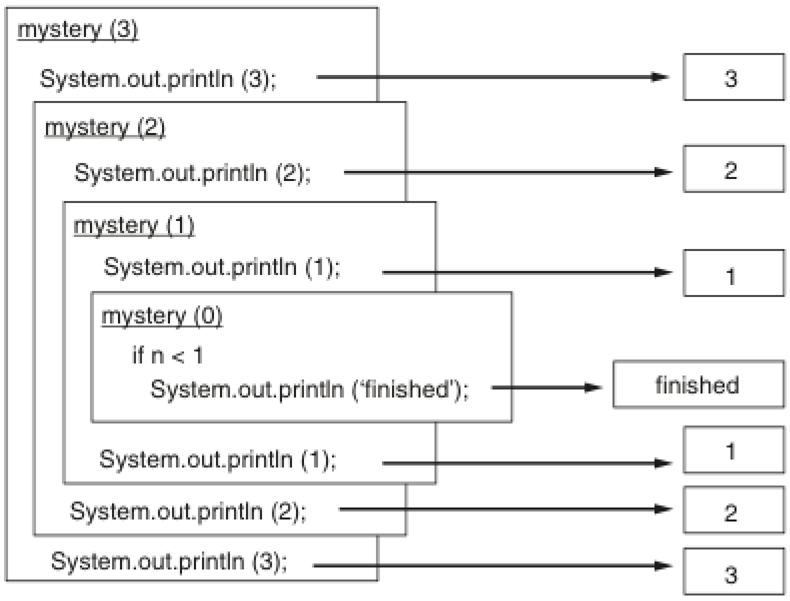
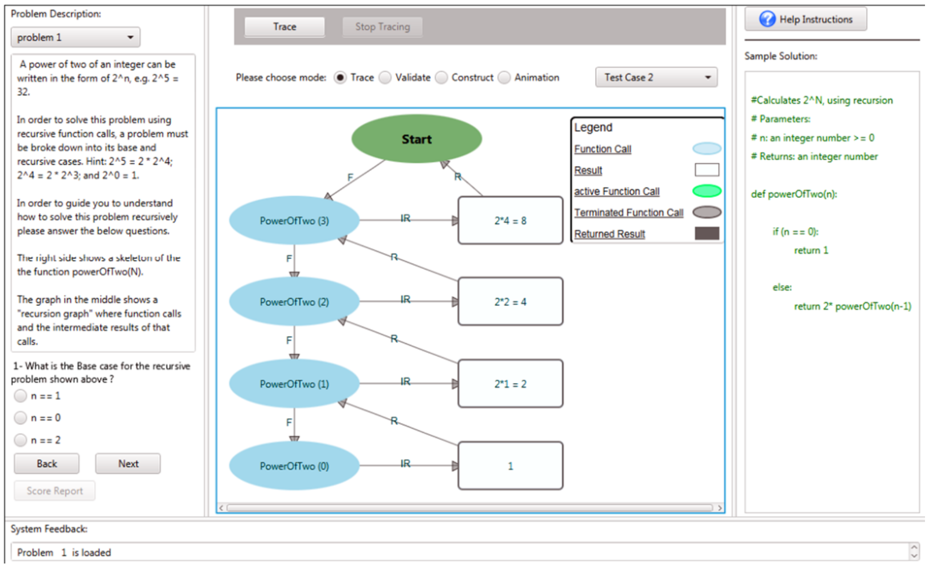


Figure 2.1: Visualization of the top-down frames method. [5]

chose to use more visual activities, such as the classic Tower of Hanoi, Sierpinkski's Carpet, and the Borax box example. What was important is that the frustrations of teaching and understanding recursion to younger children was similar to those of undergraduate learners. [4]

AlZoubi, Fossati, and others created a tool named ChiQat-Tutor to teach recursion both visually and conceptually. Most important is the visualization using a graph, seen in figure 2.2. With the graph, they were able to trace, validate, construct, and animate the recursive process. They were also able to answer questions given to them within the program. Attempting to bridge the gap between the algorithmic, conceptual parts of recursion and the actual execution of the computer, it improved learning and understanding by a small margin. Seeing the recursive process physically, learners can understand how the recursive code executes on the computer. [2] We can see that, even though recursion is both a difficult topic to teach for educators and to learn for students, as educators, we can help by visualizing the recursive process.



Problem Description:

problem 1

A power of two of an integer can be written in the form of 2^n, e.g. 2^5 = 32.

In order to solve this problem using recursive function calls, a problem must be broke down into its base and recursive cases. Hint: 2^5 = 2 * 2^4; 2^4 = 2 * 2^3; and 2^0 = 1.

In order to guide you to understand how to solve this problem recursively please answer the below questions.

The right side shows a skeleton of the the function powerOfTwo(N).

The graph in the middle shows a "recursion graph" where function calls and the intermediate results of that calls.

1- What is the Base case for the recursive problem shown above ?

- n == 1
- n == 0
- n == 2

Please choose mode: [✓ Trace] Validate Construct Animation

Sample Solution:

```
#Calculates 2^N, using recursion
# Parameters:
# n: an integer number >= 0
# Returns: an integer number

def powerOfTwo(n):

    if (n == 0):
        return 1

    else:
        return 2* powerOfTwo(n-1)
```

System Feedback:

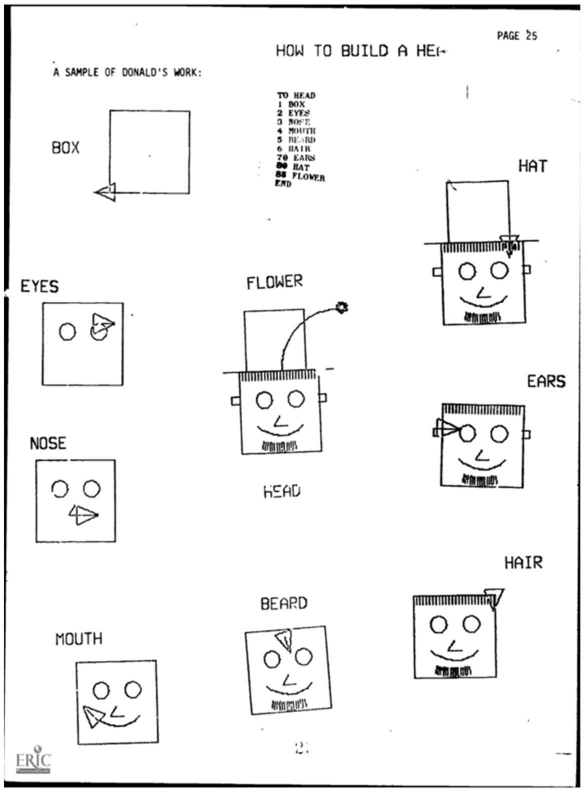
Problem 1 is loaded

Figure 2.2: The ChiQat-Tutor Program. [1]

2.2 The Turtle as a Pedagogical Tool

The programming turtle originated as a part of Papert and other's LOGO programming language. It was designed as a pedagogical exercise to teach various programming concepts. In their 1978 interim report, they outline what each of the various kids did when using the turtle. They discussed the value of LOGO, as it was easy to "mess about", easy to repeat exercises, and easy to try the "obvious."[8] As the years went on, the turtle became not just a tool for teaching young children, but also a tool for teaching undergraduate students as well. Caspersen and Christensen discussed a Java implementation of the turtle, and their use of it in an introductory computer science course. Adding onto Papert's original LOGO turtle, they use the turtle to introduce concepts of objects and classes, allowing for a natural use of multiple turtles.[3] From this, we can see that the turtle is a staple pedagogical tool for introductory programming. It is well tested, and has lived as a tool throughout the years.

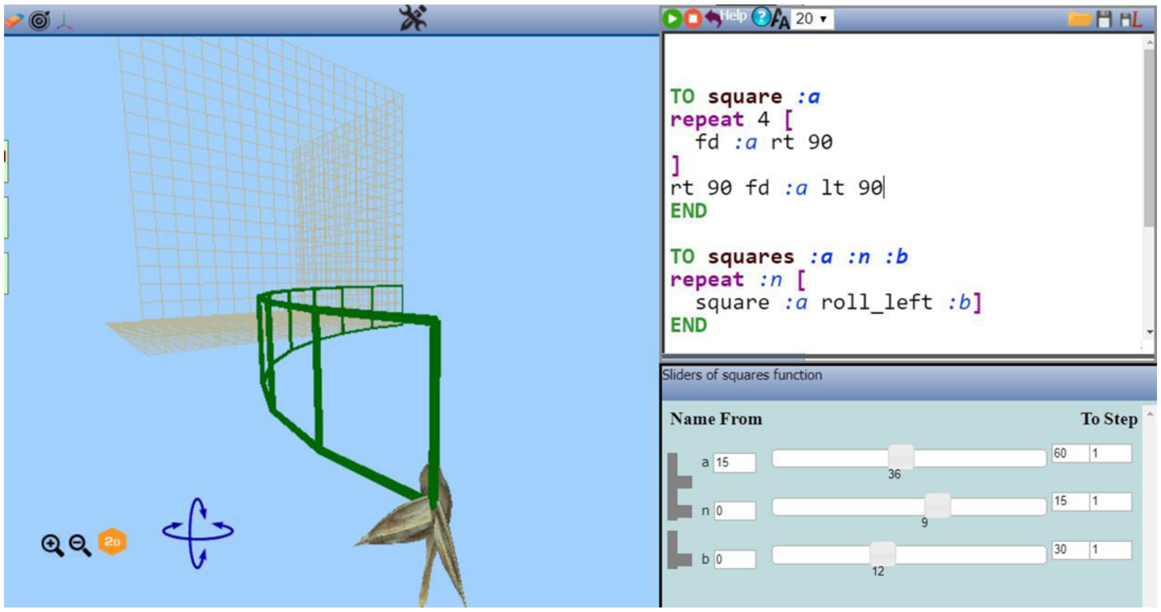
There have been various attempts at creating a 3D version of the turtle. Kynigos and Grizioti developed a LOGO dialect that was only the turtle, and fully 3D, seen in figure 2.4. They also created various UI elements such as sliders and numeric input



```
TO HEAD
1 BOX
2 EYES
3 NOSE
4 MOUTH
5 BEARD
6 HAIR
70 EARS
80 HAT
85 FLOWER
END
```

Figure 2.3: Various drawings done by a sixth-grade student using the turtle. [10]

fields to modify variables at runtime. They concluded that, while their students were able to apply computational skills and meaningfully reason about 3D space well, the program did not explicitly improve understanding of iteration or conditionals. Most of the computational skills that were shown were mostly during the part of the activity where the students were able to work on their own personal drawing. This brings up the notion that there should be more open-ended, and creative activites for this turtle. [6] Currently, there is only one 3D programming turtle in VR. This is a VR game for the PC, developed in 2020 by Wyand. [11] It uses block style programming, similar to Scratch[9]. Since this was a commercial game, and not explicitly a pedagogical tool, there is no research on whether this is an effective teaching tool for introductory programming.



```
TO square :a
repeat 4 [
  fd :a rt 90
]
rt 90 fd :a lt 90
END

TO squares :a :n :b
repeat :n [
  square :a roll_left :b]
END
```

Sliders of squares function

Name	From		To	Step
a	15	36	60	1
n	0	9	15	1
b	0	12	30	1

Figure 2.4: 3D turtle developed my Kynigos and Grizioti. [6]

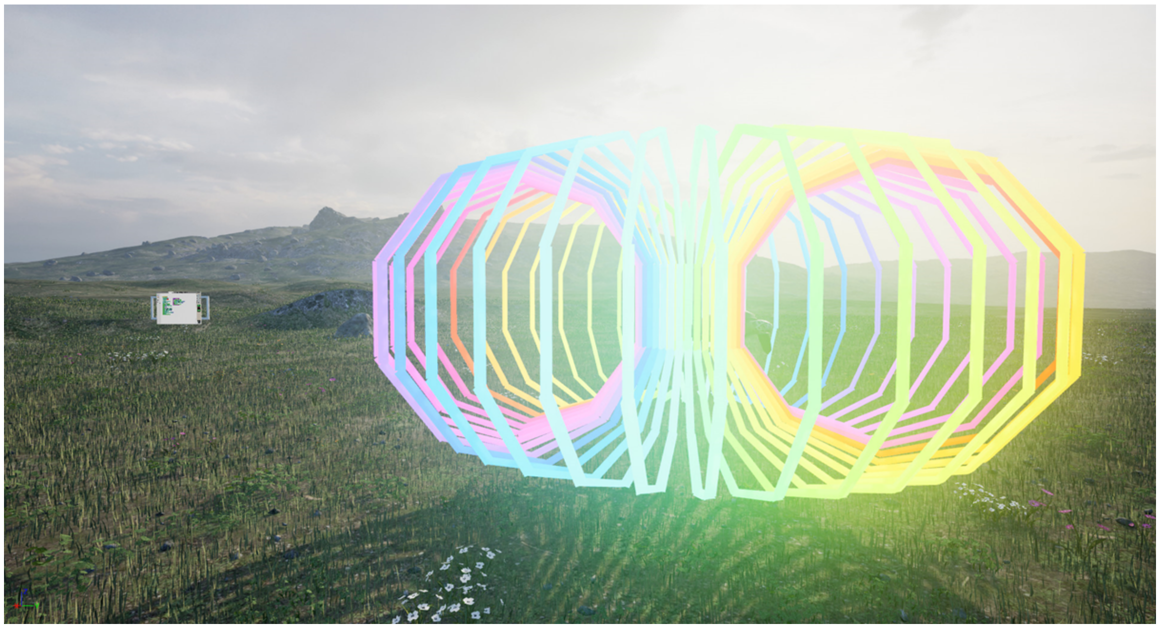


Figure 2.5: Screenshot of TurtleVR from the Steam store page. [11]

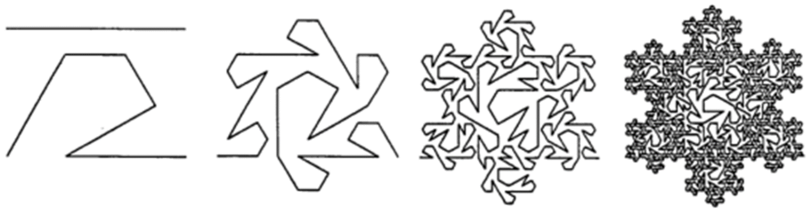


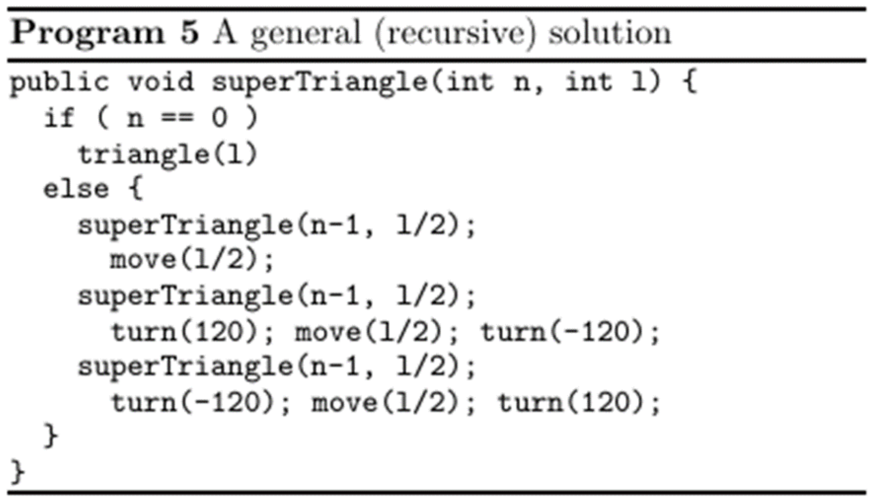
Figure 2.6: A snowflake fractal made using the turtle. [7]

2.3 Turtles All the Way Down

Using the turtle to teach recursion is very common, specifically using fractals. In 1987, Liss and McMillan taught recursion using the turtle. Specifically, they taught recursion using various fractals, such as the Koch snowflake, and other fractal snowflake designs. This encouraged students to experiment and get creative with mixing the turtle with other various technologies. [7] This experimentation allowed the students to understand recursion easier, as it was a tangible visualization of the recursive process.

Caspersen and Christensen used their Java turtle to teach recursion as well. One of their main examples was recursively creating Sierpinkski's triangle. They focused on the concept of parameterization, and passing a recursive parameter to a function. Students did create the program fairly easily, given little guidance after having the concept first explained, then shown to them. [3]

While the 2D version of the turtle has been widely used as a pedagogical tool for recursion, none have used a 3D version of the turtle. This may be due to the difficulty of managing rotations in 3D, compared to 2D. By adding complex 3D rotations, it makes understanding the topic of recursion, an already difficult topic, more difficult to understand. Similarly, there have been no studies on using a VR turtle for pedagogy. Given that TurtleVR, the game by Wyand, is the only VR turtle made at the moment, pedagogical research with a VR turtle is unlikely to exist.



Program 5 A general (recursive) solution

```
public void superTriangle(int n, int l) {
  if ( n == 0 )
    triangle(l)
  else {
    superTriangle(n-1, l/2);
      move(l/2);
    superTriangle(n-1, l/2);
      turn(120); move(l/2); turn(-120);
    superTriangle(n-1, l/2);
      turn(-120); move(l/2); turn(120);
  }
}
```

Figure 2.7: The code to create Sierpinkski's triangle in Java. [3]

Chapter 3

Approach

To answer the question, "How does teaching with a 3D/VR Turtle benefit or hinder a student's ability to understand recursion compared to the two-dimensional (2D) Turtle," we developed a study to compare the pedagogical effects of a 2D turtle and a 3D/VR turtle. First, we developed the general outline of what the students were going to do in the study. Students would first take a pre-survey and pre-quiz. The pre-survey would determine the background knowledge and confidence the student had about various topics, especially recursion. The pre-quiz would then actually determine what the student knew beforehand about recursion specifically. Students would then complete a series of activities with the turtle. Half the students would do a series of activities designed with the 2D turtle, while the other half would complete activities with the 3D/VR turtle. Finally, the students would complete a post-quiz and post-survey. The post-quiz would be similar to the pre-quiz, asking the same amount of questions and hitting similar topics. The post-survey was a more general survey, asking students to discuss their confidence in recursion and their enjoyment of the activities.

```java
static void squareRecursive(Turtle2D turtle, double length, int i) {
    // Base Case
    if (i >= 4) {
        return;
    }
    // Recursive Case
    turtle.move(length);
    turtle.clockwise(90);
    squareRecursive(turtle, length, i + 1);
}
```

Figure 3.1: The recursive version of a square function that the students previous saw.

3.1 Activity Design

Before the actual study with students, we first needed to create the activities, quizzes, and survey the students would be doing. We first created a series of activities that the students would do as a course in recursion, using the 2D turtle. Each activity would focus on a topic of recursion.

For example, let's consider the first major section of the activity. The student is first introduced to the definitions of recursion, the base case, and the recursive case. They are then given a snippet of code that converted a previous iterative solution to a recursive solution, seen in figure 3.1. They are then asked to convert another function that they wrote previously to a recursive solution, similar to this function. Importantly, we teach a few things:

1. The definition of recursion.
2. The base case and the recursive case.
3. Iteration and recursion are both looping structures.
4. You can convert between iteration and recursion.

Each activity keeps introducing various topics of recursion with more examples and exercises. In the later activities, we touch on:

```
static void Tree(
    Turtle2D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.color(0, 1.0 / (maxHeight - height), 0);
    turtle.move(length);
    turtle.right(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(0, 1.0 / (maxHeight - height), 0);
    turtle.left(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(1.0 / (maxHeight - height), 0, 0);
    turtle.right(angle);
    turtle.move(-length);
}
```

Figure 3.2: Code for visualizing depth in a Tree in 2D.

1. The concept of recursive depth.

2. The function stack, and how it is managing the recursive process.

3. The skill of tracing the inner path of computation within a recursive function.

Critically, we attempted to visualize these concepts using the 3D turtle. Take the `Tree` functions from both the 2D and 3D versions. This is a classic turtle program to illustrate recursion and recursive depth. To visualize the recursive depth in 2D, students are asked to color the branches of the tree based on the depth, seen in figure 3.2. To visualize the recursive depth in 3D, we instead gave the tree depth using 3D space, seen in figure 3.3.

```
static void Tree(Turtle3D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.yaw(angle);

    turtle.move(length);
    turtle.pitch(-angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle);
    turtle.move(-length);

    turtle.yaw(-angle);
}
```

Figure 3.3: Code for visualizing depth in a Tree in 3D.

3.2 The Turtle Program and Client

After creating the activities, we developed a 3D turtle that could be viewed in VR to best fit all the activities. The 3D turtle was developed in the Godot Game engine, as it was open-source, and had VR support. The turtle program hosts a TCP server to receive JSON data. This allows it to communicate over a JSON API, letting developers create a client in their preferred language. [1] In our study, we used Java, as it is the language used in introductory programming courses at our institution. [2] We can see the results of the tree programs in figures 3.4 and 3.5.

In the Java client, students are either given the Turtle2D class, or the Turtle3D class. The Turtle2D class follows closely to the original Papert Turtle, seen in fig-

[1]The turtle program can be seen at `https://github.com/mrSun421/turtle`.

[2]The turtle Java client used can be seen at `https://github.com/mrSun421/turtle_java_client`.

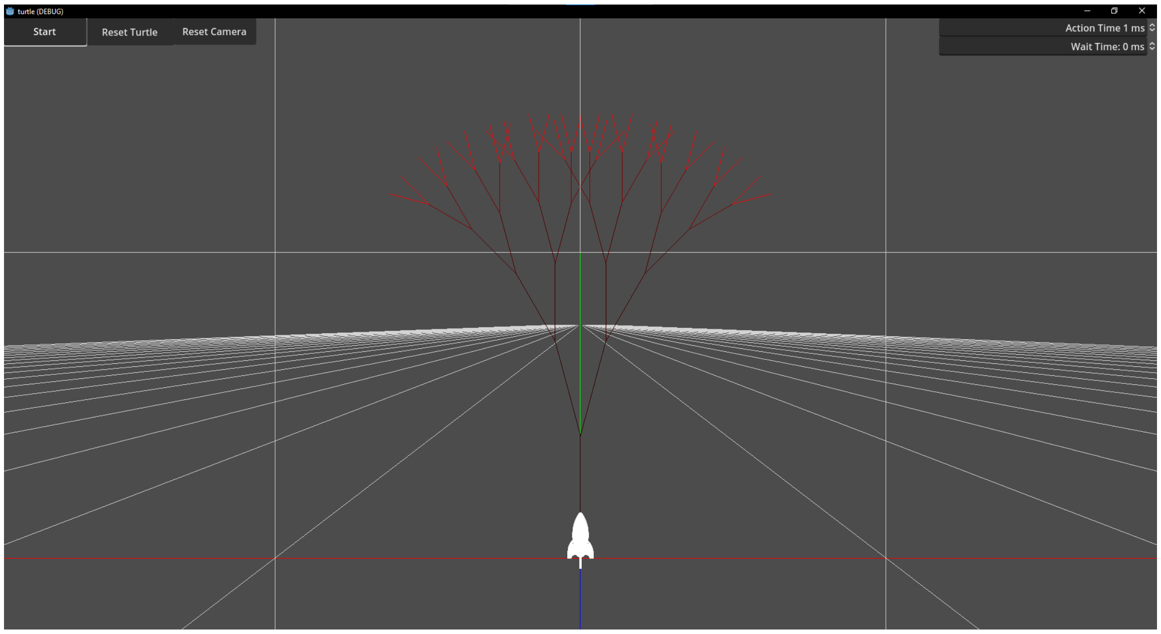


Figure 3.4: Result of the code seen in figure 3.2.

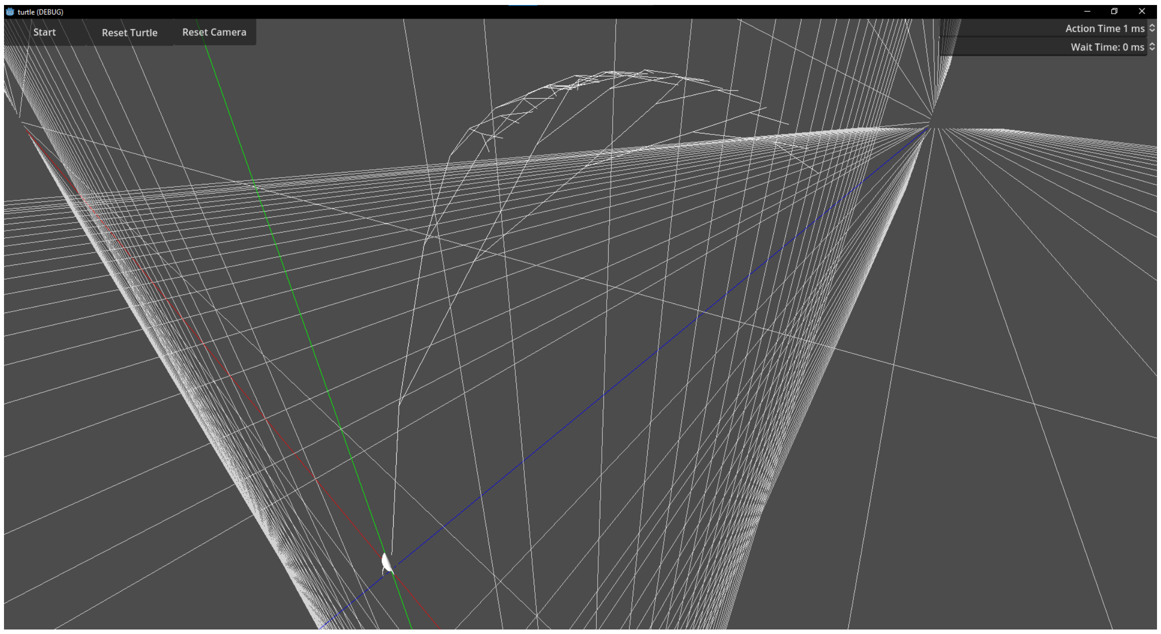


Figure 3.5: Result of the code seen in figure 3.3.

ure 3.6. Similarly, the Turtle3D class has the same functions except for rotations, seen in figure 3.7. Each class holds an inner TurtleClient, which is the class sending and receiving data over TCP. For example, move(5.0) from Turtle2D calls the function sendMoveAction(5.0) which is part of TurtleClient. sendMoveAction sends {'move': 5.0} as JSON to the turtle program. Turtle3D will call the same function that is part of TurtleClient in its move function. After the JSON data is sent to the turtle program, it converts the JSON into an action that the turtle can interpret. Each action is added to a queue of actions. Since the turtle program does not know when the client is finished, the student needs to start the turtle by either calling start in the client, or hitting the start button in the turtle program. After starting the turtle, it will continue to execute each action in the queue in order until the queue finishes or is reset by the student. For the desktop version, students were able to control their camera by dragging the mouse while holding middle click. In VR, students are able to control their position, rotation, and scale by gripping the controllers and dragging themselves.

The programming workflow starts by opening the turtle program. The student then creates a Java program using one of Turtle2D or Turtle3D. The student then run the program, sending each command called using JSON over TCP. To start, the student then either calls start at the end of their Java program, or hits the start button in the turtle program to have the turtle execute the actions. After seeing their results, they can continue to code in the Java client.

```
public class Turtle2D {
    private TurtleClient turtleClient;

    public Turtle2D();

    public void penUp();
    public void penDown();
    public void move(double length);
    public void left(double degrees);
    public void right(double degrees);
    public void clockwise(double degrees);
    public void counterclockwise(double degrees);
    public void clear();
    public void start();
    public void color(double r, double g, double b);
}
```

Figure 3.6: Simplified interface for the Turtle2D class.

```
public class Turtle3D {
    private TurtleClient turtleClient;

    public Turtle3D();
    public void penUp();
    public void penDown();
    public void move(double length);
    public void yaw(double degrees);
    public void pitch(double degrees);
    public void roll(double degrees);
    public void clear();
    public void start();
    public void color(double r, double g, double b);
}
```

Figure 3.7: Simplified interface for the Turtle3D class.

3.3 Surveys and Quizzes

Next, we developed surveys and quizzes for students to take before and after the activities. The pre-survey focused on their overall confidence and understanding in recursive knowledge. They were asked to rank various statements on a 5-point Likert scale from ”strongly disagree” to ”strongly agree.” Some statements asked include: I am comfortable using VR; I am confident in using 3D-based programs; I can quickly understand unfamiliar topics; etc. The post-survey asked questions about enjoyment of activities, as well as general confidence in understanding.

The quizzes focused on each specific topic of recursion, similar to the activities.

Consider the following function.

```
f():
    for i = 0; i < 10; i += 1 {
        print(i)
    }
```

Convert this for loop into a recursive function. The outline of the function is given to you.

```
f():
    fRecursive(0)

fRecursive(x):
    // Fill in this if statement.
    if          :
        return
    // Recursive code below.
    fRecursive(           )
```

Figure 3.8: Question 1 of the pre-quiz.

Each quiz is 5 questions long, and was designed to get more difficult with each question. Importantly, the quiz tested the topics with general code as opposed to turtle specific code. Consider figure 3.8. In the question, there is no turtle specific code, and only asks about numbers. However, it does touch upon the topic of converting between an iterative solution and a recursive solution, something discussed earlier in the activities.

To determine whether the students improved, we would compare between the pre-quiz and post-quiz, as well as qualitatively checking to see if there were any differences between the 2D and 3D groups. We would also look at whether various qualities seen in the surveys impacted how the students went about completing the activities and quizzes. For example, if a student was used to using 3D programs, they might complete the activities faster than other students. Similarly, if a student was not confident about recursion, they might struggle on the pre-quiz. Looking for patterns like these were essential to this study.

Chapter 4

Experiments and Analysis

A total of seven college computer science students completed the study. Four students completed the 2D version of the study and three completed the 3D/VR version of the study.

The students generally took about two hours to complete everything. The surveys took the least amount of time, as it only asked very basic questions. Students took the most time on the activities, completing it in about an hour. Interestingly, students took a longer amount of time on the quizzes than expected. We expected students to take about 15-20 minutes to complete the quizzes. Instead, they took about 25-30 minutes to complete each quiz. When students became frustrated or confused, we allowed them to ask questions about the activities/quizzes. Some students needed clarification on certain parts, such as definitions of terms, or usage of certain functions during the quizzes. Feedback was given after all the activities and quizzes.

Students did not have difficulty following along with the activities. As mentioned earlier, most of the questions asked while completing the activities were simply for clarification. The students were able to code and complete each activity without much trouble. This was expected, as the activities were designed to be done without guidance as if they were given these in a textbook or as homework. While doing the activities, a lot of time was spent waiting for the programs to finish. Because of the way the turtle client was designed, each command to the turtle was being sent one-by-one over the internet (TCP). This is really slow, and especially bad with recursion, as many of the fractals created with recursion usually take exponential time. For example, one student made the depth for the tree, the maxDepth of the code in figure 3.2, larger than intended. It took a substantial amount of time, both sending the commands and executing them. We can tell from both the VR usage and the slow client that there was difficulty in interacting with the turtle. Students became visibly disengaged when they were not actively engaging with the content. This may have hampered understanding, but it is unclear by how much.

For both versions, students did enjoy using the turtle, and the 3D/VR version did generally have favorable reviews. This may be due to the novelty of VR, as some 3D/VR students never used VR before. However, for the 3D/VR versions, some did state that removing and placing the headset to write and debug code was annoying. Due to the limitations of VR technology, coding in VR is impractical. Students had to place and remove the headset to go between the turtle and coding. They did clarify, however, that it was fun to see the turtle move around. One student was peculiar in their VR usage. This student was familiar with using VR. Interestingly, the student used the visible space between the headset and the nose to code, as opposed to removing and putting on the headset, which the other VR students did.

Considering the low sample size, we cannot come to sweeping conclusions about the effectiveness of VR on education. However, even with a low sample size, we can see certain patterns emerge.

Students understanding of recursion did improve. However, comparing the quiz results between the 2D and 3D/VR, there does not seem to be any significant difference in understanding. For the pre-quiz, students were able to answer questions 1-3, but struggled on questions 4 and 5. On the post-quiz, students were able to confidently

Turtle Student's Quiz Results

		Pre-quiz					Post-quiz				
	Student ID	q1	q2	q3	q4	q5	q1	q2	q3	q4	q5
2D	0	✓	✓	✓	●	●	✓	✓	✓	✓	●
	1	✓	✓	✓	✗	✗	✓	✓	✓	●	●
	2	✓	✓	✓	✓	●	✓	✓	✓	✓	●
	3	✓	✓	✓	●	●	✓	✓	✓	✓	✗
3D	4	✓	✓	●	✗	✗	✓	✓	✓	✓	●
	5	✓	✓	✓	✗	●	✓	✓	✓	●	●
	6	✓	✓	✓	●	●	✓	✓	✓	●	✓

Table 4.1: Table of results of each quiz. Note that ✓ means they got it correct, ● means they partially got it correct, and ✗ means they got it wrong/did not answer.

Turtle Student's Quiz Results (Numeric)

		Pre-quiz			Post-quiz		
	Student ID	✓	●	✗	✓	●	✗
2D	0	3	2	0	4	1	0
	1	3	0	2	3	2	0
	2	4	1	0	4	1	0
	3	3	2	0	4	0	1
3D	4	2	1	2	4	1	0
	5	3	1	2	3	2	0
	6	3	2	0	4	1	0

Table 4.2: Numeric table of results of each quiz. This is a counting version of table 4.1

answer questions 1-4, but still made mistakes on question 5. The results were similar for both the 2D and 3D/VR groups. If one were given only the pre-quiz and post-quiz, they would not be able to tell whether the student completed the 2D or 3D/VR activities. The 3D/VR activities did not have substantial impact on understanding for it to be noticeable.

The most common mistake was determining what values to return in the function. Some would only return a portion of the recursive value, instead of the whole value. For example, the final question in the post-quiz asked the students to create a recursive function that reverses a string. The mistake that students made was to return only the recursive function call, and not appending the last character to the start, seen in

Student ID →	0	1	2	3	4	5	6
I have a good understanding of computer science skills in general.	4	4	4	4	4	4	5
I have a strong understanding of recursion.	3	3	4	3	4	3	3
I have a strong understanding of 2D space (translation, rotation, etc.).	3	5	3	3	3	3	4
I have a strong understanding of 3D space (translation, rotation, etc.).	3	5	2	2	3	3	2
I have a strong mathematical background.	3	3	5	3	3	3	4
I can quickly understand unfamiliar topics.	4	4	4	4	3	3	4
I have good problem-solving skills.	5	5	5	5	3	3	5
I am confident in using 3D-based programs (Blender, AutoCAD, etc.).	2	5	3	2	3	4	1
I am comfortable using VR.	4	3	3	3	3	4	1

Table 4.3: Results from the surveys on a 5-point Likert scale. 1 is "Strongly Disagree", while 5 is "Strongly Agree."

```
f(word):
    return fRecursive(word, len(word))

fRecursive(word, length):
    if length == 0:
        return ""
    return fRecursive(word, length - 1)
```

Figure 4.1: Example of the common error students made.

figure 4.1. This could be due to the nature of the turtle program and activities. Since the students were issuing commands instead of values to the turtle, they never return any values in functions. Having to determine actual values were never addressed in the activities.

Interestingly, there was a technical split between those who were familiar with 3D programs and VR and those who were not. For those that were already confident with using 3D programs, they generally completed the activities faster than those who did not. Those students seemed to grasp more complicated geometrical concepts faster, such as rotation in 3D. The students that were not familiar with 3D programs did take longer. However, understanding on recursion was not harmed, since the quizzes did not ask any questions about the turtle itself. This could indicate that the 3D rotation may not hinder understanding as much as previously thought. The students were able to consolidate their understanding of recursion and apply it to more abstract questions.

Chapter 5

Conclusion

Due to low sample size, no sweeping and complete conclusions can be made at the moment. Given the surveys and quizzes, we can say that students did improve their understanding. However, as of the moment, we see no obvious benefits for using 3D/VR over 2D other than enjoyment. If one were to look at the quizzes in isolation, they would not be able to tell whether they completed either the 2D or 3D/VR activities. Fortunately, we did not see any harm in the study either. The 3D/VR turtle did not make the student's understanding worse.

There are many ways to improve upon this study. The two categories of improvements are technical improvements and design improvements. First, the technical improvements may increase usability. One technical issue was with the slow client for the turtle. Better networking coding techniques, such as buffers and asynchronous requests may solve the speed issue.A large amount of feedback from the 3D/VR students was the difficulty in using the program. Recall that, when coding, students had to repeatedly put on and take off the headset. This is because text in VR is difficult to read unless it is large. There is also the problem of blocking vision to the keyboard and mouse. For education purposes, we cannot assume that a student knows how to type or use a mouse without vision. This may be rectified with the advent of

Augmented Reality (AR). If students were able to have both a 3D turtle viewable, while being able to code on a computer, it may reduce the frustrations with usability. These technical improvements reduce the time spent waiting. Learning only occurs when students are actually engaging with the content. By reducing downtime, we may let the student understand more in a shorter time.

Next, the study's design can be improved. Clearly, a larger sample size would improve the study. Expanding the scope of the study to not only include recursion, but other topics such as iteration and conditionals would also be beneficial. Another method is splitting the students even further into 3 groups instead of just 2: 2D, 3D, and VR. This would determine whether the usability problem is from the extra complexity added from introducing another dimension, or it is a limitation of VR itself. Even though there were hints that the complexity of 3D does not impact understanding, we could not conclude for sure. By dividing the study in this way, we can more confidently make that statement. Another improvement is the approach taken with the activities. The activities could incorporate more 3D techniques, such as more complex 3D fractals or creating polyhedra from the turtle. However, it is not known whether this would complicate understanding more compared to simplifying the activities. More time must be invested into the study to carefully develop and tune a series of activities that use 3D and VR to its fullest potential.

Overall, while there are no definite results, there are many clear and straightforward methods to improve the study. Students did improve as a whole, and using the 3D/VR turtle did not prevent understanding compared to the 2D turtle. We do not see, however, a tangible benefit using the 3D/VR turtle either. A future study would need to improve both the turtle program itself, the work around the turtle, and the base design of the study itself.

Appendix A

Appendix

A.1 Turtle Activities (2D)

Introduction to the Turtle

Look at `Turtle2D.java`. The following is an explanation of the various functions.

move	moves length units the direction the turtle is facing. Length can be negative, moving backwards.
left/right/clockwise/counterclockwise	rotates degrees in that direction.
penUp/penDown	enable/disable drawing of lines.
clear	clears the current queue of actions for the turtle. It is recommended you always do this at the top of the program to remove previous requests to the turtle.
start	starts the queue of actions on the turtle.
color	colors the line using rgb. The color values are between 0.0-1.0.

Example A.1.1: Square

Let's draw a square. We should move forward a length, rotate 90 degrees, and repeat that 4 times. It should look like this:

```
static void square(Turtle2D turtle, double length) {
    for (int i = 0; i < 4; i++) {
        turtle.move(length);
        turtle.clockwise(90);
    }
}
```

Now, try this problem!

Problem A.1.1: Polygon

Create a function using a for loop that draws a regular polygon with n sides. The function name and parameters have been given to you.

```java
static void polygon(Turtle2D turtle, int n, double length) {
    double angle = 360.0 / n;
    for (int i = 0; i < n; i++) {
        turtle.move(length);
        turtle.clockwise(angle);
    }
}
```

Introduction to Recursion

Definition A.1.1: Recursion

An algorithmic technique of solving a problem where the solution depends on solutions of smaller versions of the problem.

Think of, for example, a picture inside a picture inside a picture. Another example would be two mirrors pointing at each other. At some point, there's something stopping it from continuing on forever. In recursion, we separate these ideas of self-similary and stopping with two cases.

Definition A.1.2: Base Case

In a recursive algorithm, the case where the algorithm stops. It is when the solution to the problem is known/trivial. Stops the algorithm from infinitely looping.

Definition A.1.3: Recursive Case

In a recursive algorithm, the case that calls itself. It is needed to create smaller sub-problems when calling itself.

Example A.1.2: Square

Consider the square function from earlier. We stop when `i >= 4`. That is our base case. The recursive case is drawing the line, then rotating. We then increase `i` by 1.

```java
static void squareRecursive(Turtle2D turtle, double length, int i) {
    // Base Case
    if (i >= 4) {
        return;
    }
    // Recursive Case
    turtle.move(length);
    turtle.clockwise(90);
    squareRecursive(turtle, length, i + 1);
}
```

We then call this function with `squareRecursive(turtle, length, 0)`.

Problem A.1.2: PolygonRecursive

We are going to create a recursive version of the polygon function.

1. Identify the base case. In other words, why and when do we stop in the loop?

2. Identify the recursive case. As a hint, consider we've already moved forward and turned clockwise once. What has changed after doing so? Can you think of a smaller sub-problem?

3. Create the recursive functions given the partially filled out functions.

```java
static void polygon(Turtle2D turtle, int n, double length) {
    polygonRecursive(turtle, n, length, 0);
}


static void polygonRecursive(
    Turtle2D turtle,
    int n,
    double length,
    int i) {
    // Base Case
    if (i >= n) {
        return;
    }
    // Recursive Case
    double angle = 360.0 / n;
    turtle.move(length);
    turtle.clockwise(angle);
    polygonRecursive(turtle, n, length, i + 1);
}
```

Tree and Recursive Depth

Definition A.1.4: Recursive Depth

The number of recursive calls that are waiting for the inner recursive function to finish. This is also the height of the stack of function calls.

Example A.1.3: Tree

Write the following function in your code.

```
static void Tree(Turtle2D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.move(length);
    turtle.right(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.left(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.right(angle);
    turtle.move(-length);
}
```

Run the code in main as the following:

```
Tree(turtle, 4.0, 15, 1, 0)
```

. Try different `maxHeight` values.

In example A.2.3, notice how our recursive depth is our height in this case. We only increase the height whenever we call another recursive function. When we finish the inner function, we return to the caller function, letting our current height reduce by 1 again. It's a lot easier to see in the next example.

Example A.1.4: Tree

Modify the tree function that it now adds in color.

```
static void Tree(
    Turtle2D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.color(0, 1.0 / (maxHeight - height), 0);
    turtle.move(length);
    turtle.right(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(0, 1.0 / (maxHeight - height), 0);
    turtle.left(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(1.0 / (maxHeight - height), 0, 0);
    turtle.right(angle);
    turtle.move(-length);
}
```

What should happen is that the ends of the tree should be a brighter color of green compared to the lower parts of the tree, which should be darker. This should visualize the depth even clearer in this case.

Notice how we seem to only be executing the most recently called function. On the computer, internally, it keeps track of which functions you have called. This is called the function stack.

Definition A.1.5: Function Stack

A stack of functions to be called and executed. While a function is executing, if it calls another function, then that function is pushed onto the stack, and then begins executing.

If we have a recursive function, generally the function stack's height is the recursive depth (if no other functions are called). Note that this is the stack that is referred to in a stack overflow error. The function stack gets too large, and the program errors out!

Problem A.1.3: TripleTree

Modify the tree function such that instead of drawing to the left and right, it also draws in the center. In other words, it should recursively draw a tree to the right, middle, and left, instead of only the right and middle.

```
static void tripleTree(
    Turtle2D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.color(1.0 / (maxHeight - height), 0, 0);
    turtle.move(length);
    turtle.right(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(1.0 / (maxHeight - height), 0, 0);
    turtle.left(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.left(angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.color(1.0 / (maxHeight - height), 0, 0);
    turtle.right(angle);
    turtle.move(-length);

}
```

Koch

Problem A.1.4: Koch's Snowflake

Write this function in your code.

```
static void koch(
    Turtle2D turtle,
    double length,
    int maxDepth,
    int depth) {
    if (depth >= maxDepth) {
        turtle.move(length);
        return;
    }
    koch(turtle, length / 3, maxDepth, depth + 1);
    turtle.left(60);
    koch(turtle, length / 3, maxDepth, depth + 1);
```

```
        turtle.right(120);
        koch(turtle, length / 3, maxDepth, depth + 1);
        turtle.left(60);
        koch(turtle, length / 3, maxDepth, depth + 1);
}
```

1. Run the function with various `maxDepth` values.
2. Describe the base case. When do you stop? What actions do you take when you stop?
3. Describe the recursive case? What do we do before recursively calling again?
4. Note that we cannot use the same coloring trick to visualize the depth. Why?

Recognize that, while doing this problem, we can notice the function stack as we are running this. It will always prioritize the most recently called function over everything else.

A.2 Turtle Activities (3D)

Introduction to the Turtle

Look at `Turtle3D.java`. The following is an explanation of the various functions.

`move`	moves `length` units the direction the turtle is facing. Length can be negative, moving backwards.
`yaw/pitch/roll`	rotates `degrees` in that axis. To rotate the other way, use a negative degree value.
`penUp/penDown`	enable/disable drawing of lines.
`clear`	clears the current queue of actions for the turtle. It is recommended you always do this at the top of the program to remove previous requests to the turtle.
`start`	starts the queue of actions on the turtle.
`color`	colors the line using `rgb`. The color values are between 0.0-1.0.

Example A.2.1: Square

Let's draw a square. We should move forward a length, rotate 90 degrees, and repeat that 4 times. It should look like this:

```
static void square(Turtle3D turtle, double length) {
    for (int i = 0; i < 4; i++) {
        turtle.move(length);
        turtle.pitch(90);
    }
}
```

Now, try this problem!

Problem A.2.1: Polygon

Create a function using a for loop that draws a regular polygon with n sides. The function name and parameters have been given to you.

```
static void polygon(Turtle3D turtle, int n, double length) {
    double angle = 360.0 / n;
    for (int i = 0; i < n; i++) {
        turtle.move(length);
        turtle.pitch(angle);
    }
}
```

Introduction to Recursion

Definition A.2.1: Recursion

An algorithmic technique of solving a problem where the solution depends on solutions of smaller versions of the problem.

Think of, for example, a picture inside a picture inside a picture. Another example would be two mirrors pointing at each other. At some point, there's something stopping it from continuing on forever. In recursion, we separate these ideas of self-similary and stopping with two cases.

Definition A.2.2: Base Case

In a recursive algorithm, the case where the algorithm stops. It is when the solution to the problem is known/trivial. Stops the algorithm from infinitely looping.

Definition A.2.3: Recursive Case

In a recursive algorithm, the case that calls itself. It is needed to create smaller sub-problems when calling itself.

Example A.2.2: Square

Consider the square function from earlier. We stop when `i >= 4`. That is our base case. The recursive case is drawing the line, then rotating. We then increase `i` by 1.

```
static void squareRecursive(Turtle3D turtle, double length, int i) {
    // Base Case
    if (i >= 4) {
        return;
    }
    // Recursive Case
    turtle.move(length);
    turtle.pitch(90);
    squareRecursive(turtle, length, i + 1);
}
```

We then call this function with `squareRecursive(turtle, length, 0)`.

Problem A.2.2: PolygonRecursive

We are going to create a recursive version of the polygon function.

1. Identify the base case. In other words, why and when do we stop in the loop?
2. Identify the recursive case. As a hint, consider we've already moved forward and turned clockwise once. What has changed after doing so? Can you think of a smaller sub-problem?
3. Create the recursive functions given the partially filled out functions.

```
static void polygon(Turtle3D turtle, int n, double length) {
    polygonRecursive(turtle, n, length, 0);
}


static void polygonRecursive(
    Turtle3D turtle,
    int n,
    double length,
    int i) {
    // Base Case
    if (i >= n) {
        return;
    }
    // Recursive Case
    double angle = 360.0 / n;
    turtle.move(length);
    turtle.pitch(angle);
    polygonRecursive(turtle, n, length, i + 1);
}
```

Tree and Recursive Depth

Definition A.2.4: Recursive Depth

The number of recursive calls that are waiting for the inner recursive function to finish. This is also the height of the stack of function calls.

Example A.2.3: Tree

Write the following function in your code.

```
static void Tree(Turtle3D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.move(length);
    turtle.pitch(-angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle);
    turtle.move(-length);
}
```

Run the code in main as the following:

```
Tree(turtle, 4.0, 15, 1, 0)
```

. Try different `maxHeight` values.

In example A.2.3, notice how our recursive depth is our height in this case. We only increase the height whenever we call another recursive function. When we finish the inner function, we return to the caller function, letting our current height reduce by 1 again. It's a lot easier to see in the next example.

Example A.2.4: Tree

Modify the tree function that it now adds in depth.

```
static void Tree(
    Turtle3D turtle,
    double length,
    double angle,
    int maxHeight,
    int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.yaw(angle);

    turtle.move(length);
    turtle.pitch(-angle);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle * 2.0);
    Tree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(-angle);
    turtle.move(-length);

    turtle.yaw(-angle);
}
```

What should happen is that the tree should have depth. This should visualize the recursive depth even clearer in this case.

Notice how we seem to only be executing the most recently called function. On the computer, internally, it keeps track of which functions you have called. This is called the function stack.

Definition A.2.5: Function Stack

A stack of functions to be called and executed. While a function is executing, if it calls another function, then that function is pushed onto the stack, and then begins executing.

If we have a recursive function, generally the function stack's height is the recursive depth (if no other functions are called). Note that this is the stack that is referred to in a stack overflow error. The function stack gets too large, and the program errors out!

Problem A.2.3: TripleTree

Modify the tree function such that instead of drawing to the left and right, it also draws in the center. In other words, it should recursively draw a tree to the right, middle, and left, instead of only the right and middle.

```
static void TripleTree(Turtle3D turtle,
        double length,
        double angle,
        int maxHeight,
        int height) {
    if (height >= maxHeight) {
        return;
    }
    turtle.yaw(angle);


    turtle.move(length);
    turtle.pitch(-angle);
    TripleTree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle);
    TripleTree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(angle);
    TripleTree(turtle, length * 0.8, angle, maxHeight, height + 1);
    turtle.pitch(-angle);
    turtle.move(-length);

    turtle.yaw(-angle);
}
```

Koch

Problem A.2.4: Koch's Snowflake

Write this function in your code.

```
static void koch(
        Turtle3D turtle,
        double length,
        int maxDepth,
        int depth) {
    if (depth >= maxDepth) {
        turtle.move(length);
        return;
    }
    turtle.yaw(90);
    turtle.move(length);
    turtle.yaw(-90);
    koch(turtle, length / 3, maxDepth, depth + 1);
    turtle.pitch(60);
    koch(turtle, length / 3, maxDepth, depth + 1);
    turtle.pitch(-120);
    koch(turtle, length / 3, maxDepth, depth + 1);
    turtle.pitch(60);
    koch(turtle, length / 3, maxDepth, depth + 1);


    turtle.yaw(90);
    turtle.move(-length);
    turtle.yaw(-90);
}
```

1. Run the function with various `maxDepth` values.
2. Describe the base case. When do you stop? What actions do you take when you stop?
3. Describe the recursive case? What do we do before recursively calling again?
4. Note that we cannot only use the `yaw` function to visualize depth; We have to move in and out as well. Why?

Recognize that, while doing this problem, we can notice the function stack as we are running this. It will always prioritize the most recently called function over everything else.

A.3 Pre-Quiz Questions

Q1

Consider the following function.

```
f():
    for i = 0; i < 10; i += 1 {
        print(i)
    }
```

Convert this for loop into a recursive function. The outline of the function is given to you.

```
f():
    fRecursive(0)


fRecursive(x):
    // Fill in this if statement.
    if             :
        return
    // Recursive code below.



    fRecursive(             )
```

Q2

Consider the following function.

```
f(x):
    if x <= 0:
```

```
3               print(x)
4               return
5         padding = ""
6         for i = 0; i < x; i++ {
7               padding += "a"
8         }
9         printf("%s%d",padding, x)
10        f(x-1)
```

Which lines are the base case? Which lines are the recursive case? What will this function output if $x = 3$?

Q3

Consider the following function.

```
f(x):
      if x <= 0:
            return
      if x % 2 == 0:
            f(x/2)
      else:
            f(x-1)
      print(x)
```

This function outputs the following when $x = 7$:

```
1
2
3
6
```

7

Modify f such that it instead prints out the following when $x = 7$:

7

6

3

2

1

Hint: no new code is added. We are only moving around code.

Q4

Suppose you are given a sorted array, and you want to do binary search. The following is an iterative version of binary search. Assume that the value that you are looking for is guaranteed to be in the array for simplicity.

```
f(array, value):
    left := 0
    right := len(array) - 1
    while left <= right:
        middle := (left + right) / 2
        if value == array[middle]:
            return middle
        else if value < array[middle]:
            right = middle - 1
        else:
            left = middle + 1
```

Convert this iterative version of binary search into a recursive version. You can assume that the searching value is guaranteed to be in the array.

Q5

Create a function that finds the sum of the digits of a number using a recursive solution. You are guaranteed a non-negative integer as input (0, 1, . . .). The outline of the function is given below.

```
f(x):
    // Fill in this if statement.
    if           :
        return 0
    // Recursive code below.
```

A.4 Post-Quiz Questions

Q1

Consider the following function.

```
f(n):
    value := 1
    for i:= 1; i < n; i++ {
        value *= i
    }
    return value
```

Convert this for loop into a recursive function. The outline of the function is given to you. You are guaranteed a non-negative integer as input (0, 1, . . .).

```
 f(x):
    // Fill in this if statement.
    if           :
        return 1
    // Recursive code below.
```

Q2

Consider the following function.

```
1 fib(x):
2     if x == 0:
3         return 0
4     if x == 1:
5         return 1
6     return fib(x - 1) + fib(x - 2)
```

Which lines are the base case? Which lines are the recursive case? What will this function output if $x = 3$?

Q3

Consider the following function.

```
f(x):
    if x <= 0:
        return
    if x % 10 == 0:
        f(x/10)
    else:
        f(x-1)
    print(x)
```

This function outputs the following when $x = 103$:

```
1
10
100
101
102
103
```

Modify f such that it instead prints out the following when $x = 103$:

```
103
102
101
100
10
1
```

Hint: no new code is added. We are only moving around code.

Q4

The following is a function with a recursive function as a helper. The function returns a boolean value (true or false). Convert the function to an iterative solution.

```
f(word):
    wordLen := len(word)
    return fRecursive(word, 0, wordLen - 1)


fRecursive(word, start, end):
    if start == end:
        return true
    if word[start] != word[end]:
        return false
    if start < end + 1:
        return fRecursive(word, start + 1, end - 1)
    return true
```

Q5

Create a function that reverses a string. The outline of the function is given below.

```
f(word):
    return fRecursive(word, len(word))


fRecursive(word, length):
    // Fill in this if statement.
    if           :
        return ""
    // Recursive code below.
```

A.5 Pre-survey Questions

Each statement was asked on a 5 point Likert scale from ”Strongly disagree” to ”Strongly Agree.”

- I have a good understanding of computer science skills in general.
- I have a strong understanding of recursion.
- I have a strong understanding of 2D space (translation, rotation, etc.).
- I have a strong understanding of 3D space (translation, rotation, etc.).
- I have a strong mathematical background.

- I can quickly understand unfamiliar topics.
- I have good problem-solving skills.
- I am confident in using 3D-based programs (Blender, AutoCAD, etc.).
- I am comfortable using VR.

A.6 Turtle JSON API

The Turtle VR program uses a TCP server over localhost on port 21327 (may be changed in source code) to receive actions for the turtle. The following is a specification of what JSON data should be sent for specific turtle actions.

```
// penup or pendown
{ "action": "penup" | "pendown" }

// moves the turtle 'number' forward
{ "action": "move", "value": number }

// rotates the turtle 'number' degrees by given action
{ "action": "yaw" | "pitch" | "roll", "value": number }

// colors the line of the turtle.
// 'number' is from 0.0 to 1.0 inclusive.
{
    "action": "color",
    "color": { "r": number, "g": number, "b": number }
}

// clears the current queue of actions to be taken
{ "action": "clear" }
```

```
// starts the turtle on it's queue of actions
{ "action": "start" }
```

A.7 Source Code

The source code for the Turtle Program can be found at `https://github.com/mrSun421/turtle`. The source code for the Turtle client written in Java can be found at `https://github.com/mrSun421/turtle_java_client`.

Bibliography

[1] Omar AlZoubi, Davide Fossati, Barbara Di Eugenio, and Nick Green. ChiQat-Tutor: An Integrated Environment for Learning Recursion. 2014.

[2] Omar AlZoubi, Davide Fossati, Barbara Di Eugenio, Nick Green, Mehrdad Alizadeh, and Rachel Harsley. A Hybrid Model for Teaching Recursion. In *Proceedings of the 16th Annual Conference on Information Technology Education*, SIGITE '15, pages 65–70. Association for Computing Machinery, 2015. ISBN 978-1-4503-3835-6. doi: 10.1145/2808006.2808030. URL `https://dl.acm.org/doi/10.1145/2808006.2808030`.

[3] Michael E. Caspersen and Henrik Bærbak Christensen. Here, there and everywhere - on the recurring use of turtle graphics in CS1. In *Proceedings of the Australasian Conference on Computing Education*, pages 34–40. ACM, 2000. ISBN 978-1-58113-271-7. doi: 10.1145/359369.359375. URL `https://dl.acm.org/doi/10.1145/359369.359375`.

[4] Katherine Gunion. Curing recursion aversion — Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education, 2024. URL `https://dl-acm-org.proxy.library.emory.edu/doi/10.1145/1562877.1562919`.

[5] Orit Hazzan, Noa Ragonis, and Tami Lapidot. *Guide to Teaching Computer Science: An Activity-Based Approach.* Springer International Publishing, 2020.

ISBN 978-3-030-39359-5 978-3-030-39360-1. doi: 10.1007/978-3-030-39360-1. URL http://link.springer.com/10.1007/978-3-030-39360-1.

[6] Chronis Kynigos and Marianthi Grizioti. Programming Approaches to Computational Thinking: Integrating Turtle Geometry, Dynamic Manipulation and 3D Space. 17(2):321–340, 2018. ISSN 1648-5831. URL https://www.ceeol.com/search/article-detail?id=708776.

[7] Ivan B. Liss and Thomas C. McMillan. Fractals with turtle graphics: A CS2 programming exercise for introducing recursion. In *Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '87, pages 141–147. Association for Computing Machinery, 1987. ISBN 978-0-89791-217-4. doi: 10.1145/31820.31749. URL https://dl.acm.org/doi/10.1145/31820.31749.

[8] Seymour Papert and And Others. Interim Report of the LOGO Project in the Brookline Public Schools: An Assessment and Documentation of a Children's Computer Laboratory. Artificial Intelligence Memo No. 484, 1978. URL https://eric.ed.gov/?id=ED207799.

[9] Mitchel Resnick and Brian Silverman. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children*, pages 117–122, Boulder Colorado, June 2005. ACM. ISBN 978-1-59593-096-5. doi: 10.1145/1109540.1109556.

[10] Cynthia J. Solomon. Teaching young children to program in a LOGO turtle computer culture. 12(3):20–29, 1978. ISSN 0163-5735. doi: 10.1145/964041.964045. URL https://dl.acm.org/doi/10.1145/964041.964045.

[11] David Wyand. Turtle VR, 2020. URL http://www.gnometech.com/games/turtle-vr/.