**Decision Trees**

**CMSC435**

**Version 2.1**

**Authors:**

**Guanhua Liu**

**Lucas Cheng**

**Ameen Ali Ateeqi**

**Hugo Wang**

**Yonatan Zvi Hersh Margulies**

**Dustin Cordell Brashear**

**David Eitan Kaplan**

**Gregory John Segala**

**Table of Contents**

1. **Delivery Manifest 2**
   1. Abstract ….……………………………….………………………………… 2
   2. Deliverables .……………………………………………..………………… 3
   3. Product Specifications ..…………………………………..………………… 4
   4. Online Tool to Create Decision Trees ……………...…….………………… 5
   5. Grading Algorithm ………………...…………………..……………………. 6
   6. Computer Vision Imaging ……………….……………………..……………. 7
   7. Website Structure ……………………………………………..…………….. 11
   8. Dependencies …………………………………………………………......… 13
   9. Terminology ……………………….………………..………………………. 14
2. **Intellectual Property Statement 15**
3. **Acceptance Test Plan 16**
   1. Testing Assumptions ……………………………………………..…………. 16
   2. Website Testing ...…………………….…………………………………… 16
   3. Grading Testing ...…………………………………………...……………... 18
   4. Imaging Testing ……………………..………...…………………………….. 20
4. **Cost Estimate 23**
   1. Function Point Analysis ………………….…...……………………………. 23
   2. Website Cost Estimate …………………….…………...…………………… 25
   3. Grading Cost Estimate …………………….………...……………………… 27
   4. Imaging Cost Estimate …………………….…………………..…………… 28
   5. Maintenance Fee Estimate ………………………………………………...... 30
5. **Timeline 32**
6. **Project Advertisement 34**

**1. Product Definition**

**1.1 Abstract**

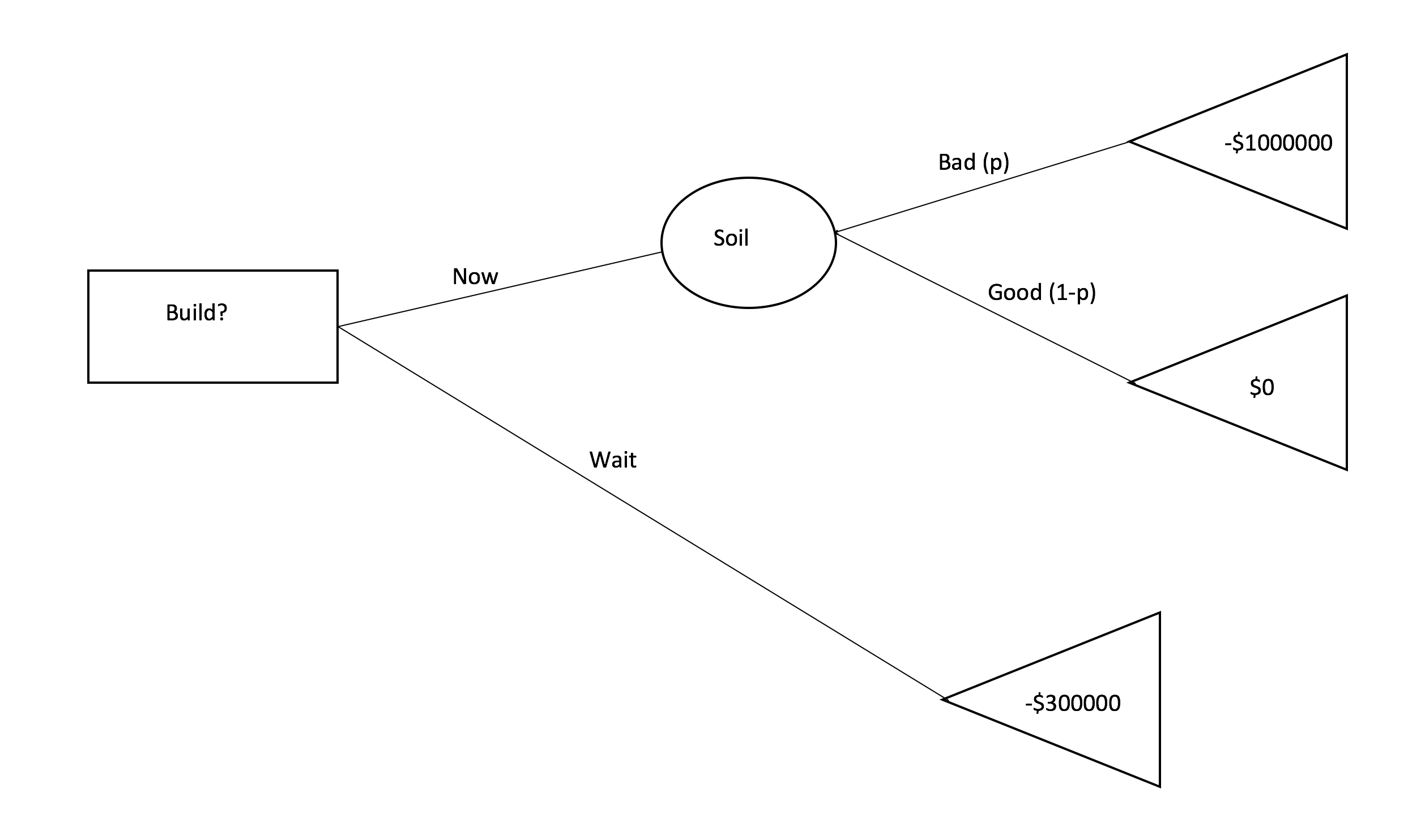
Education has developed to the point where many assignments can be graded online. Examples such as WebAssign can grade students’ responses to mathematical questions immediately after they are answered, reducing both the workload for the teacher and providing students immediate feedback on the assignment. However, while such products exist for mathematical and text-based questions, not many resources exist for grading graphical-based assignments, such as decision trees. Decision trees are a graphical representations of a series of choices or chance events that show all possible conclusions from the series of events. An example is shown in Figure 1.1

Figure 1.1

A box node is a decision node, a circle node is a chance node, and a triangle is an answer node. The probabilities with the associated chance node are noted *p* and *1-p* on the edges.

Our group of the fall 2017 class of CMSC435 is building a platform to automatically grade decision tree assignments, from both homework and exams. There will be two tools available to the instructor who uses this product: assigning decision tree homeworks via the website that students can complete online, and grading decision trees answers from exams submitted by the students. These tools are designed to not interrupt the workflow of the students and teachers and is designed for ease of use.

To meet this functionality, there are two ways that a decision tree can be created: an online tool for students to make and submit a decision tree provided by the website, and computer vision application that can read in a scanned picture of a drawn decision tree from paper. The online tool to create a decision tree will be primarily used for homework assignments, while the computer vision application will be used primarily for exams. Both of these implementations to read in a decision tree can be graded automatically using our grading algorithm.

The product will be secured by using Central Authentication System (CAS) authentication. All students and instructors need to have access to CAS in order for them to use the website. This authentication system makes the website secure, and provides a way for students and instructors to log into the website.

**1.2 Deliverables**

1. A Ruby-on-Rails website
   1. Can be accessed through internet connection
   2. Must be accessed through current version of Google Chrome, Firefox, or Microsoft Edge
   3. Can store user-input text data and XML data of decision trees
2. An online decision tree creation app (p5.js)
   1. Allows users to “draw” trees different nodes through clicking
   2. Can type for tree labels
3. Ruby grading algorithm
   1. Tells similarity to teacher created decision tree answer key
   2. Gives percentage of student’s tree being correct
4. Image processing algorithm
   1. Takes images as input
   2. Reads and stores decision trees with given input
5. Installation Guide
   1. Shows users how to set up their own server to host website
6. Project Advertisement
   1. Quick summary showcase of what our team accomplished

**1.3 Product Specifications:**

We will be delivering a Ruby on Rails product as part of making the website, and packaged automatically along with this is an SQLite3 database and HTML/Javascript functionality. The frontend UI will display all trees graphically, with shapes as nodes, lines as edges, and labels in both edges and nodes within the decision tree. In the creation of our website, we will have three main user roles for those who access our website - instructors, students, and administrators.

1.3.1 Instructors

The website will be a central tool to how an instructor grades decision trees for all of their classes. Instructors will be able to create assignments for classes, potentially be able to grade decision tree questions from exams, and be able to see each student's grade for completed assignments.

Once the instructor has logged in using CAS authentication, the instructor is forwarded to a page where there is an option to create a class. Once they click that button, the are able to create a class by adding their students SID (student ID’s). Once a class is created, they can create an assignment for the class. Using the online tool, the instructor is able to create questions where the answer is a decision tree. The instructor then creates the answer key using the online tool for creating a decision tree. More information on the tool that creates a decision tree is described in section 1.4. The assignment can have multiple questions. Once the assignment is produced with a due date, the students of the associated class can then complete the assignment. A student cannot complete the assignment after the due date. Once a student completes the assignment, the teacher will be able to see each student's answers and the grade given to that student. Since the grading is automatic, no intervention on the instructors behalf is necessary for students to receive a grade. If the instructor does not like the grade assigned to the student, the instructor will be able to change the grade to how they see fit. The instructor can also give comments to students that then can be seen by the student.

Once the computer vision tool is complete, the instructor will be able to grade decision trees from exams that the instructor assigned in class. This imaging tool will create a richer workflow for the instructor because they would be able to automatically grade decision trees on exams instead of doing it by hand. More information about the workflow, how it works, and limitations are described in section 1.6.

1.3.2 Students

Students will not be able to access the website until they are approved by their instructor (described above). Once they have access and logged in through CAS authentication, they will be forwarded to a page where they can choose which class they are in the system for. Once they click on the right class, they will see a page of assignments with an indication showing which ones they have completed. Once a student has submitted an assignment, they will not be able to do it again. If there is an assignment that they have not submitted, they go to a page where it shows the assignment. There will be a box that shows the question created by the instructor and a place where the student can create the answer decision tree. The student will create the decision tree using the online tool provided by the website. Once the student is happy with their answers to each assignment, they press the submit button to submit the assignment. Then the assignment gets processed using the automatic grading algorithm. The grade given by the algorithm is then shown on the page that shows each assignment. Explanation of how the grading algorithm works is in Section 1.5.

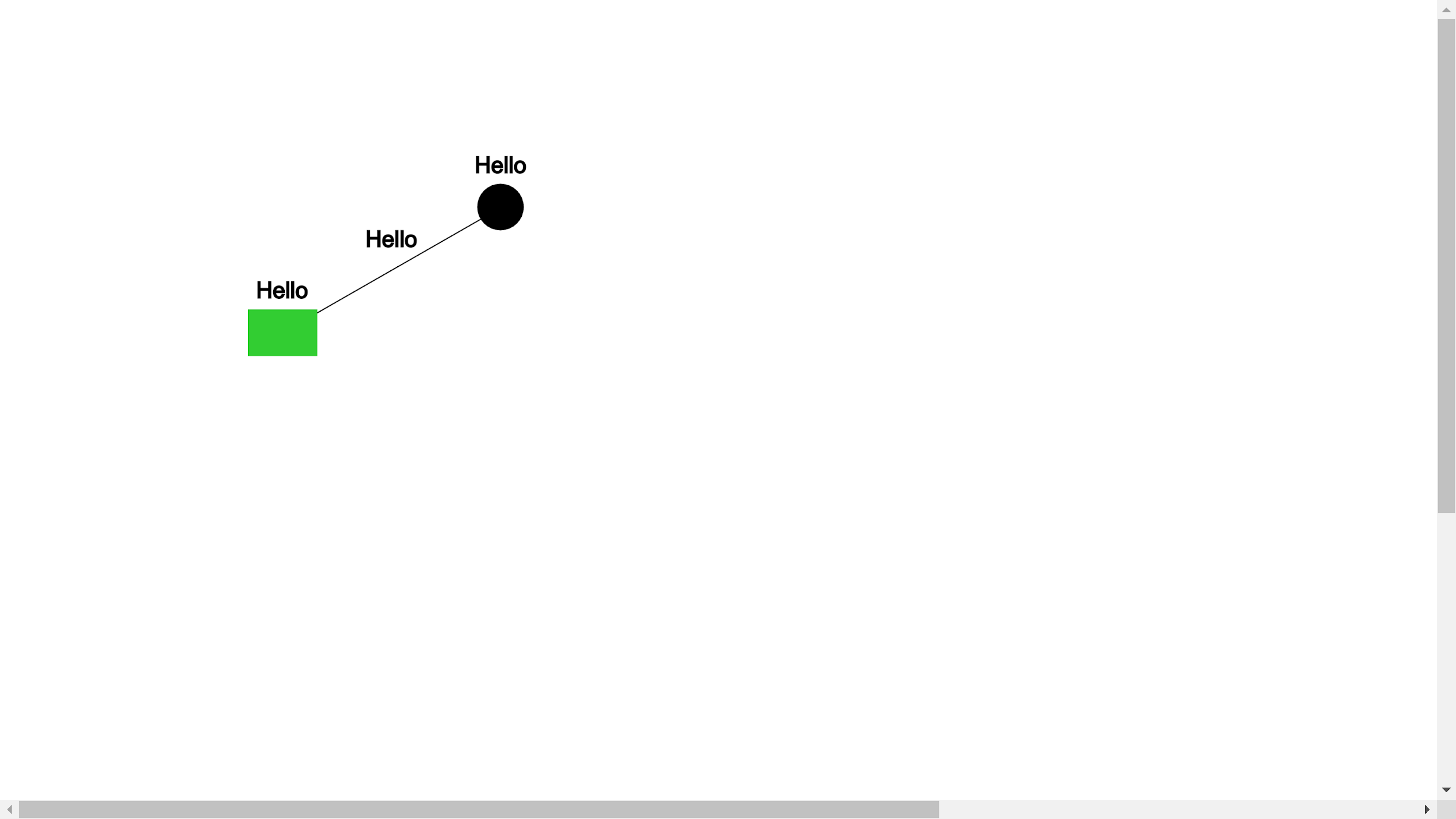
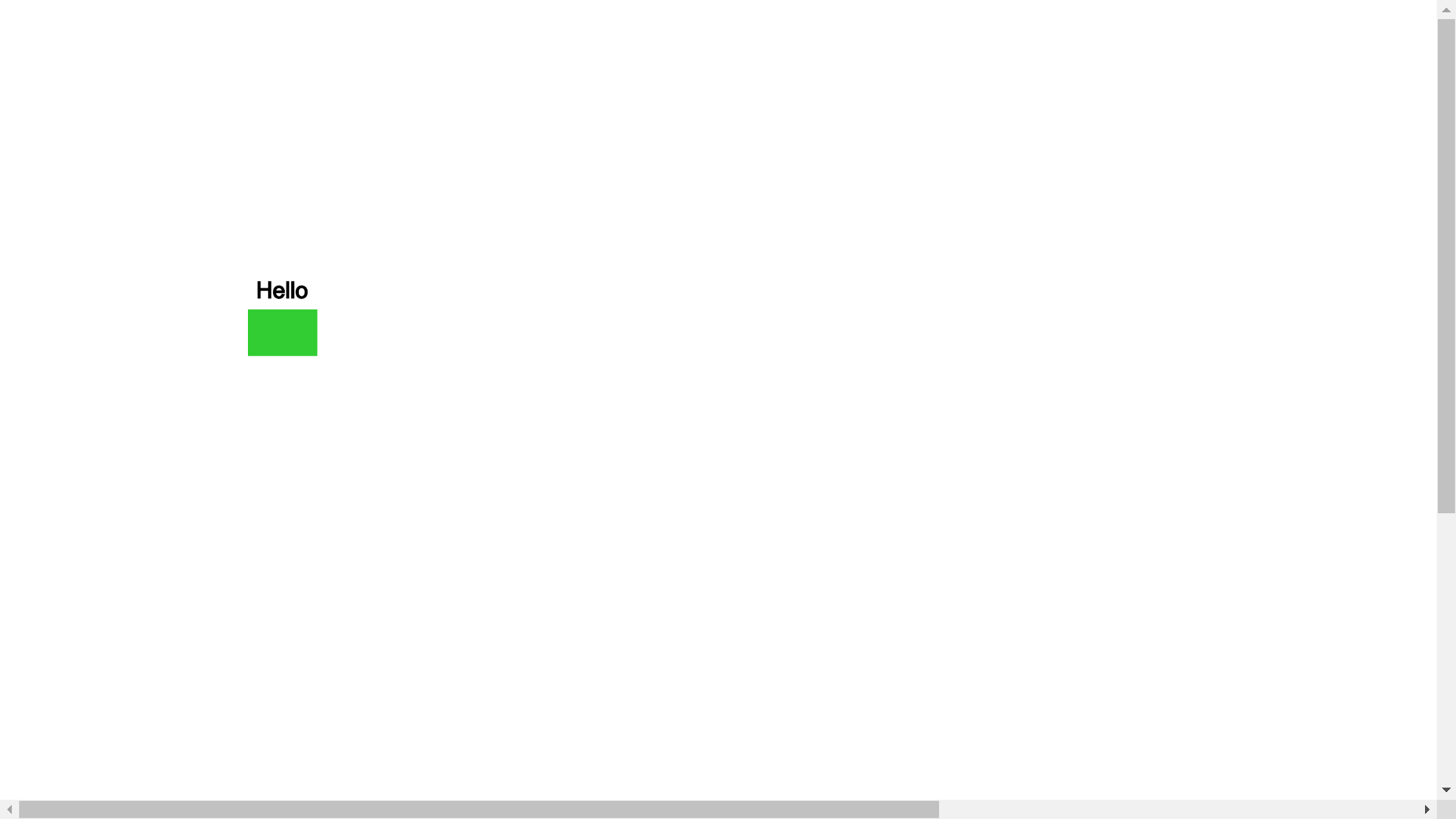
1.3.3 Administrator

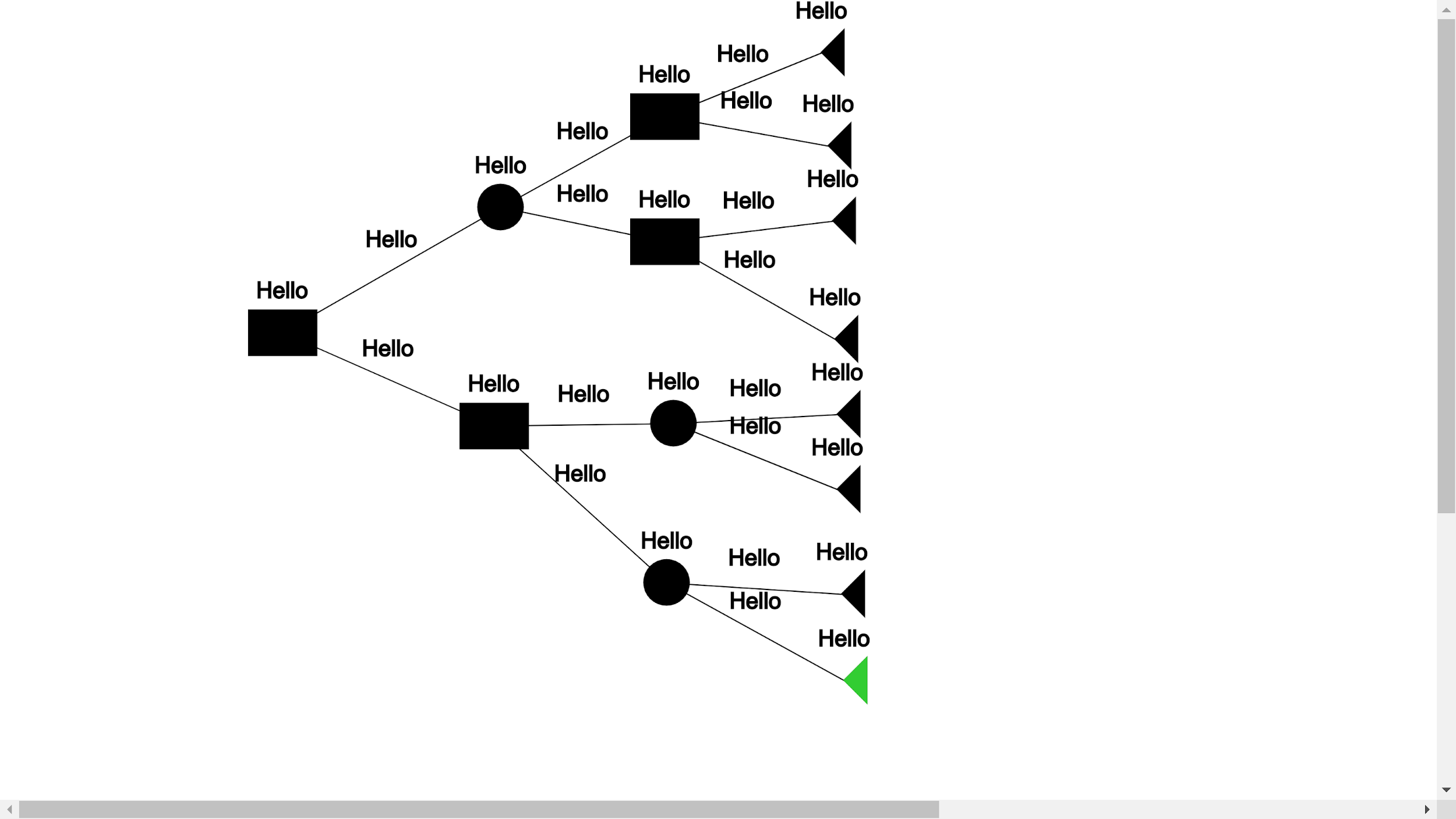
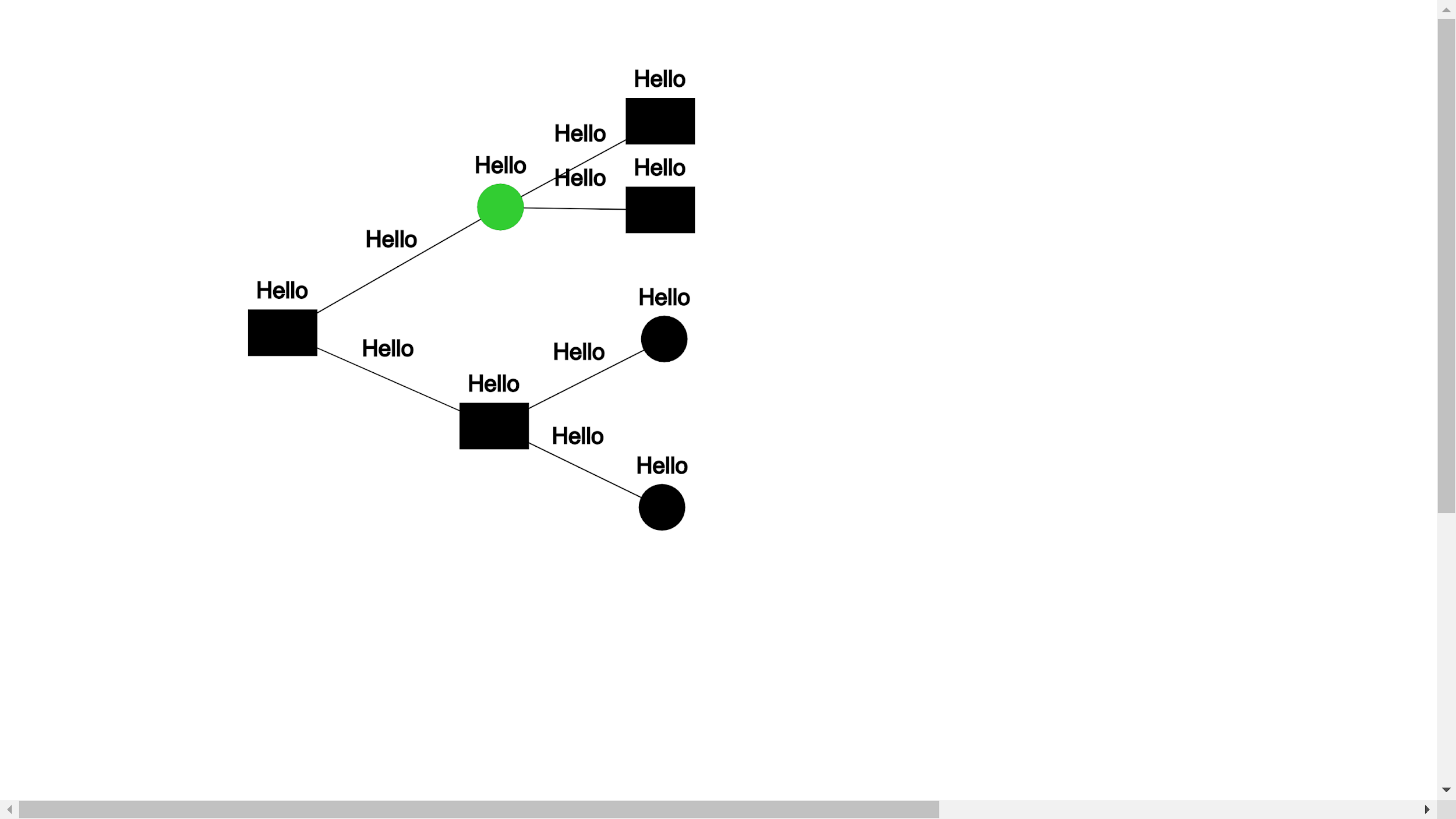
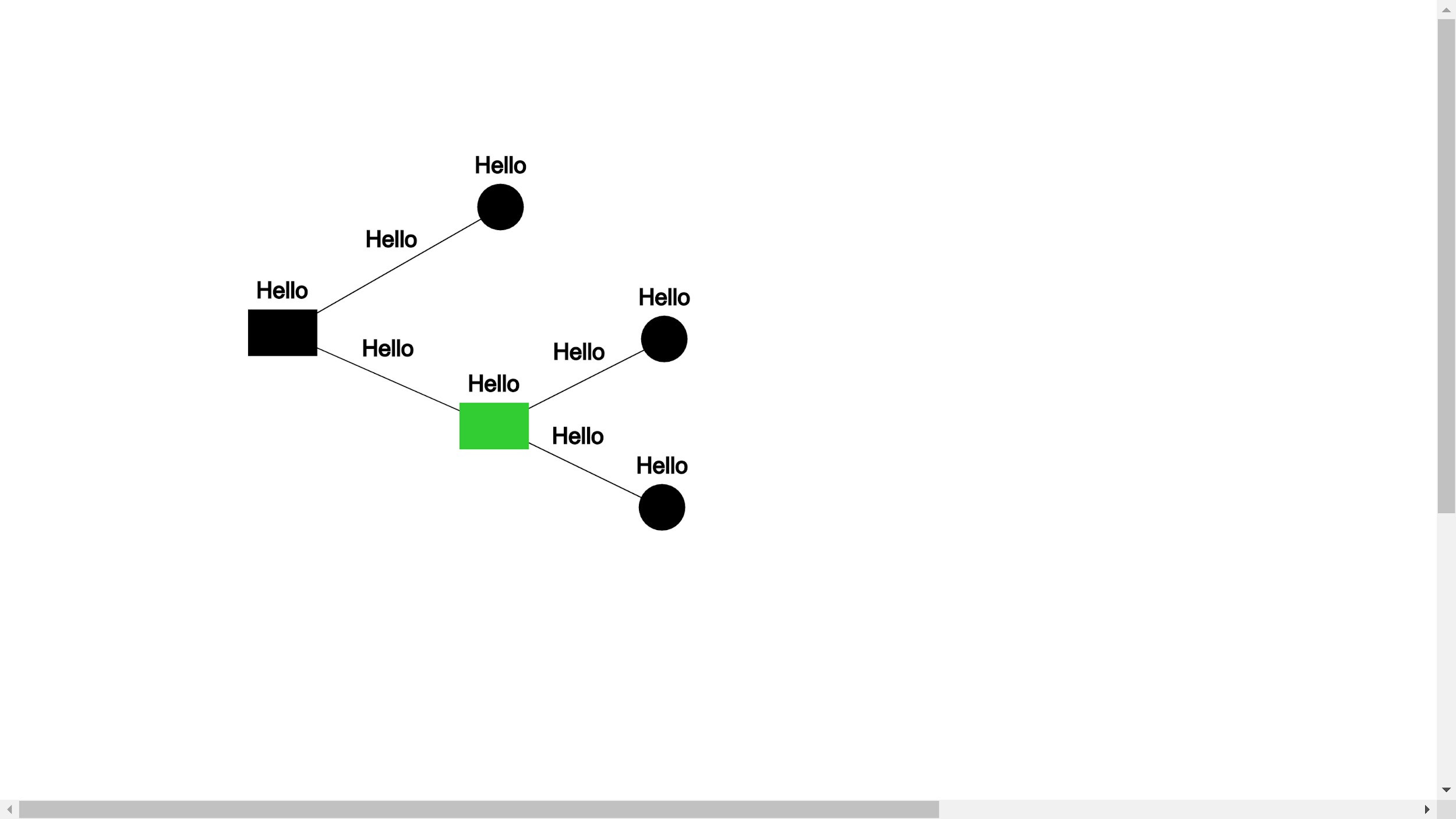
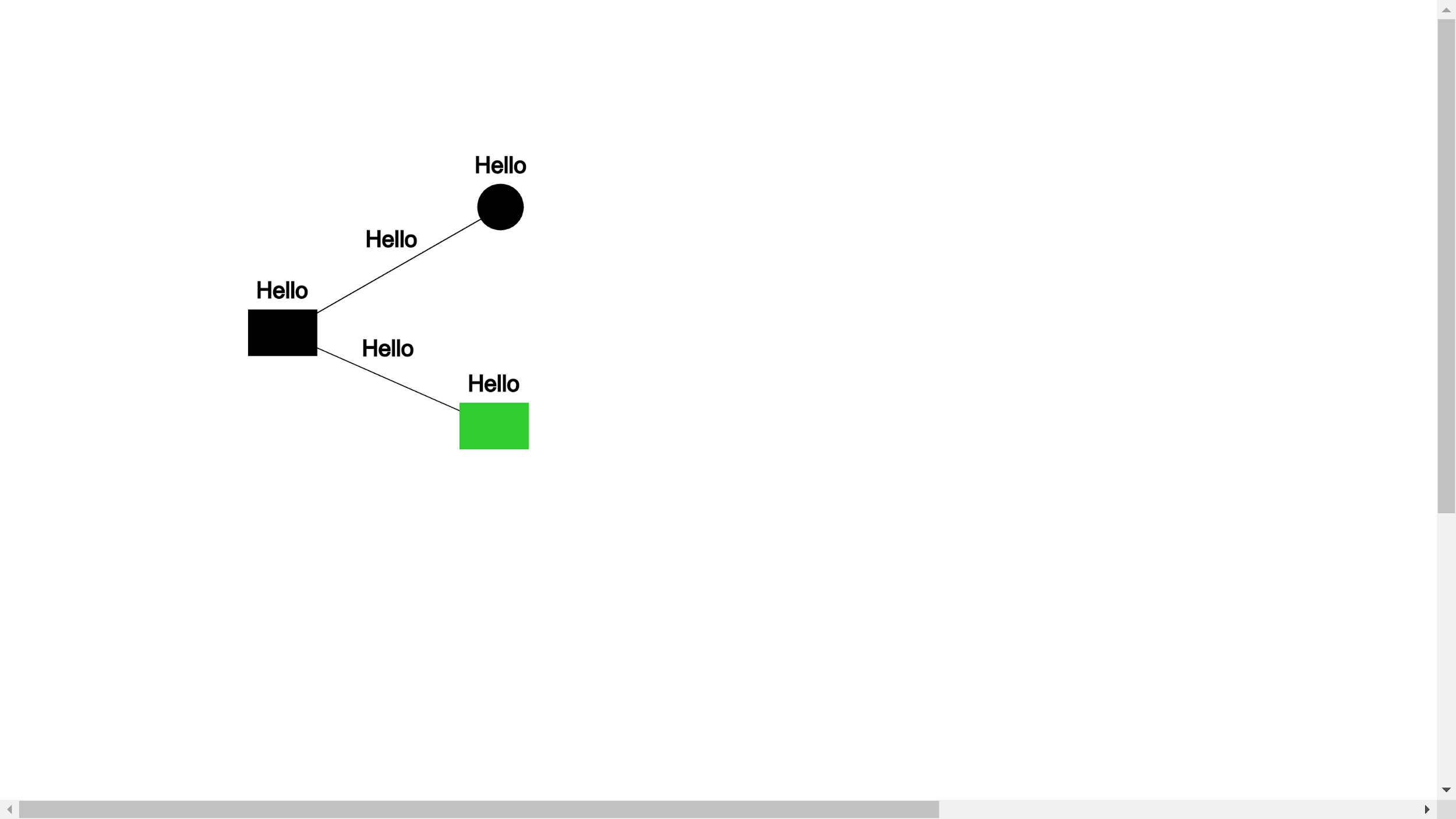
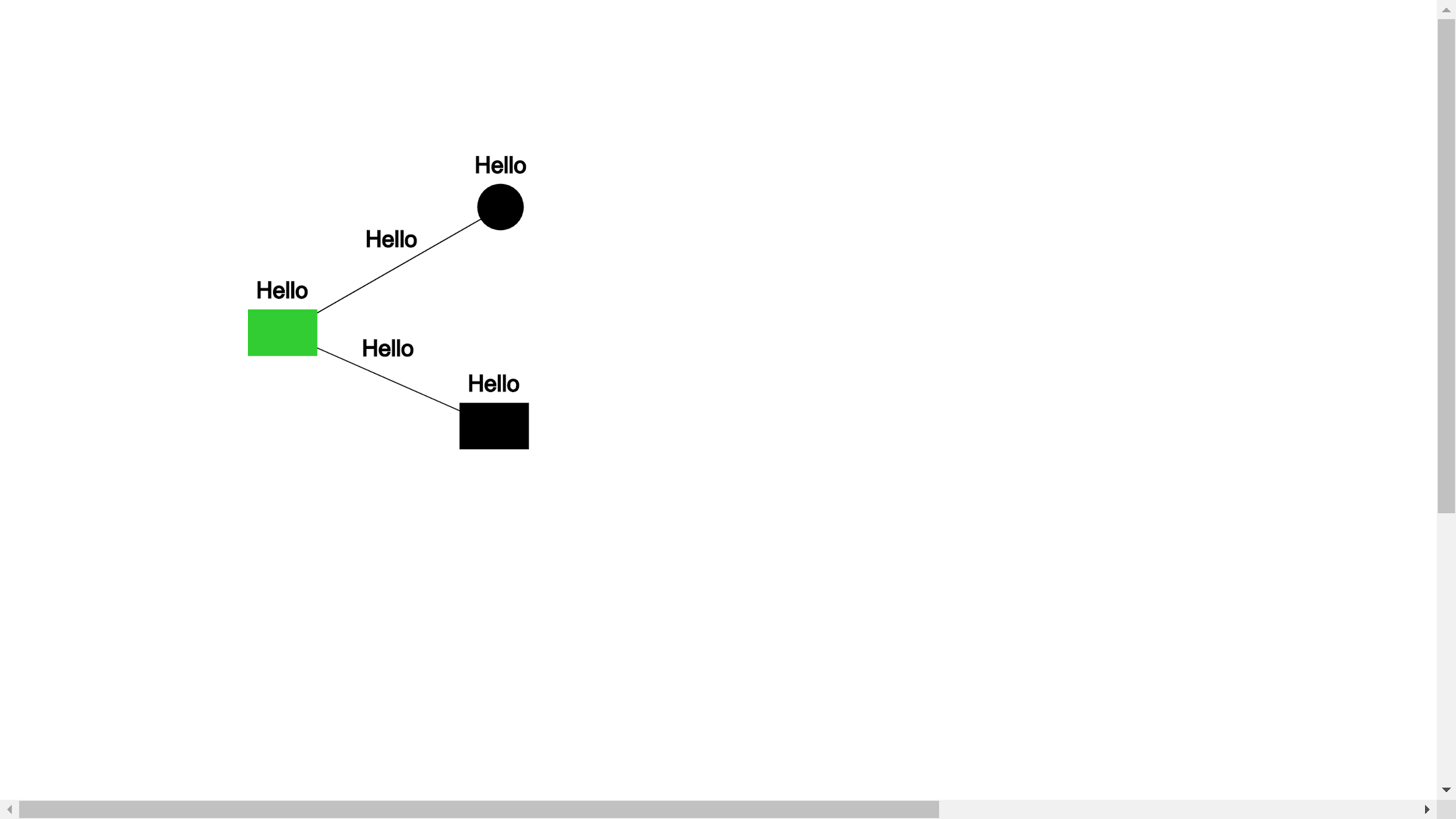
The administrator role is mostly only oversight, as this role would have the highest permissions, and would mainly only assign user roles for those who use our website (ie. they’d decide who is an administrator or an instructor). They will also have the ability to update the website, and make changes to the grading algorithm. All administrators have to have instructor level access. However, not all instructors can be administrators.

**1.4 Online Decision Tree Creation Tool:**

The specifics for how the tree creation-interactions are as follows. The online tree creation tool will be contained within its own window. The user will be able to access the tree creation window via button click whenever it is necessary to create/edit/view trees throughout the program. There will be a legend on the top right of the window that explains how to use the program as well as provide options to create different nodes -- decision, choice, or final. Users can select node types by pressing their corresponding keys: ‘D’ = decision node, ‘C’ = chance node, ‘F’ = final node. The default node type will be set to decision node. A user will be able to create the first node by mouse clicking anywhere on the window. A node of the selected type will be created at the mouse click location. After the first node is created, it will be selected and highlighted. If a node is selected, any newly created nodes will be children of the highlighted node. Lines will be draw from parent to children nodes. Clicking on any already created node will select that node and highlight it. After selecting a node, users can also choose to delete that node using the DELETE key (Note: this will delete all the children for the node), change the node’s type with the ‘V’ key, or create more children for that node by clicking on an unoccupied space within the tree creation window. The user will also be able to edit the text values of a selected node and change the edges text values from that node using interactive text boxes. All created and currently existing nodes are stored in a tree data structure which stores their type, text values, edges, and children. The tree creation tool will not allow the creation of a non-tree structure; it will not allow any nodes that have path ways back to themselves to exist and it will have one head node. The user can create trees that do not follow decision tree format (i.e. a user could create a final node and give it two chance nodes as children). At the bottom right of the screen the user will be able to click either the submit or save buttons to publish their created tree or save the tree to the database for later editing. The tree creation tool will send a submitted or saved tree to the database or other parts of the program by converting the tree data structure into a general file format (such as XML or JSON). The tree creation tool will also be able to read in the basic file format and load them to the data structure in order to display saved trees.

**Basic Tree Sketch Program Workflow:**





**Captions:**

1. To start, the user would either press “d” or click a square node icon to select a decision node, then click anywhere to create the node as shown above. Nodes are able to be labeled among creation through a textbox, and edited later
2. The user then either presses “c” or clicks a circle icon to select a chance node, and clicks within the area to create a chance node. The currently selected node is denoted by green (which is automatically the last created node), and users may click on any existing node to select it as the current selected node. Newly created nodes are children of the selected node as shown above. Edge label text is marked upon creation, and can be edited like node labels
3. The user then either presses “d” or clicks on the square node icon to create another decision child from the root.
4. The user selects a new node, which is again highlighted in green
5. User creates more children to the selected node
6. User selects a new node and creates new children to it, combing steps 4 & 5
7. User creates children to all 4 newly created nodes, making a layer of final nodes. This is done by selecting each children individually and pressing “f” or clicking the triangle icon for a final node upon creation.

**1.5 Grading algorithm**

Additionally, we will be writing Ruby scripts for our algorithm in order to grade the decision trees submitted to the website. Once given a decision tree stored in XML, it will output a percentage grade after comparing it to the answer decision tree in XML. In the grading of all trees we assume that there will be a “canonical” solution tree that the teacher will provide. Simply put - we can condense all lengthy trees into shorter trees by putting similar types of consecutive nodes into one node (See Figure 1.5 for example).

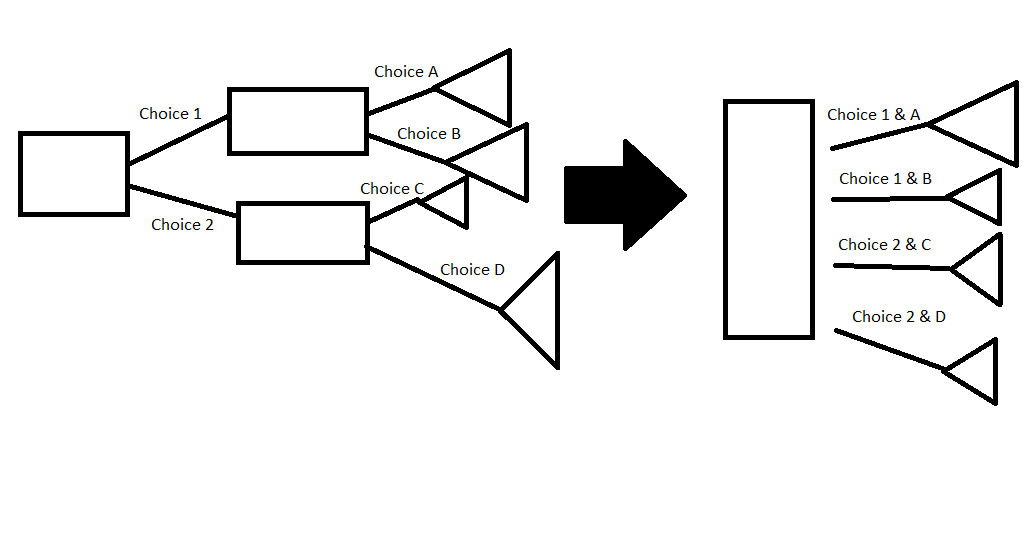


Figure 1.5

There are three main criteria that the grading algorithm will take into consideration when compared against a professor uploaded canonical solution.

1. Presence of all required nodes and edges (Percentage present from 0 to 100)
2. Similarity Estimation to canonical solution (Percentage from 0 to 100)
3. Performance (Various input -> Compare Output)

In addition to this, there will be an optional pre-processing step that will attempt to “collapse” both the canonical solution and the student tree to their respective minimal representation. The collapsed tree comparator will be a separate module within the grading algorithm. Both the regular tree comparator and the advanced collapsed tree comparator will be utilized, and the greater of the grades will be output. This step will be done for the benefit of the student as the collapsing of the tree might not necessarily increase the given grade. Along with this step the Nokogiri Ruby library will be used to parse the nested XML data into a Ruby Decision Tree object, which the algorithm will operate over. Below is a sample decision tree XML file that will be parsed into a Ruby object for comparison.

With this assumption, any tree that the students provide will be turned into a “canonical” solution (or condensed) which will then be matched. Order within the tree will not matter, and our algorithm will check the student’s answer and the correct answer between canonical versions. After checking the node structure is correct, we then make sure the labels at each node or edge is correct - allowing for truncations, misspellings and synonyms. Truncations will be taken as long as it is a subset of the larger word (eg. “p” or “prob” in place for probability). Misspellings are taken to account which accepts the word as correct as long as X% of the word is correct (where “X” is an adjustable percentage, like 70%). Synonyms are taken into account through the help of both Wordnet project, which gives a database of synonyms, and rwordnet, a ruby gem implementation that we can easily set up and use to quickly find common synonyms of given words through Ruby. Afterwards - we total up the percentage that was correct. The tree structure will be given 50% weight, and a student tree that’s converted to its canonical version that matches exactly the teacher’s answer will be given 50% of the total points. The other 50% of the points depends on the student’s answers to the labels. If all labels match the teacher’s after accounting for truncation, misspellings, and synonyms, then the student will receive another 50% of the points, giving 100% total (tree structure + labels). If the student’s answers aren’t perfectly correct, we grade and give a percentage-wise of correctness for each part, and return the final grade. For each incorrect part, we also label it and return this information labelled back on the student’s tree as part of the grading feedback as well.

The nested XML format we plan to use will have three types of nodes specified by the following format:

* <chance n = "name" e = "edge from parent to this" p = "chance to this not if CHANCE node parent" >
* <decision n = "name" e = "edge from parent to this" p = "chance to this not if CHANCE node parent"/>
* <final e = "edge from parent to this" p = "chance to this not if CHANCE node parent" v = ""/>

The name field is required for each node, while the edge field for the root node will be nil and probability for non chance nodes will also be nil.

**1.6 Computer Vision Imaging:**

Finally, we anticipate using a Python script to read in drawn decision trees using machine learning with OpenCV3 to convert simple, strictly printed trees to a digital version. Since the computer vision application is still in development stage, it is not expected to be fully functional by the initial product delivery. However, it will be improved and implemented in later updates to the software.

Since this tool of translating an image of a decision tree to a data structure is complete, it will be wrapped into a tool that allows the instructor to grade exams in a streamlined way. This will improve the workflow of the instructor because the grading process is automatic. First a description of how the software will be integrated with the website is described, then how to draw trees, then how it works, then the workflow on how to use it, and then limitations.

**1.6.1 Integration with the Website**

The computer vision algorithm will be very easily integrated with the website. The only input to the imaging algorithm is an image and the only output is a XML file of the same structure as the online tool to create decision trees. The imaging algorithm is done in Python and can be integrated with the backend using the “exec” function in Ruby. For example:

XML\_structure = exec("python3 image\_algorithm.py /location/of/Image.png")

Will call the function *image\_algorithm* passing the parameters */location/of/Image.png* with the output *XML\_structure*. If the imaging algorithm failed, the algorithm will return a string specifying the error. Thus, the Ruby backend should test the file returned to see if it is a string or not. This can be done with:

XML\_structure.is\_a?(String)

If this boolean is *True*, then it means there was an error and the imaging algorithm cannot read it. If it is *False*, the XML\_structure file is an XML file that describes the structure of the tree interpreted by the imaging algorithm in the same format that is used with the rest of the website, so the output can be accessed and used by every tool on the website.

1.6.2 Rules for drawing trees

**Color convention**: For the initial prototype, the drawing of the decision trees will look for color: red for nodes, blue for edges, green nodes indicating labels for edges. The colors are shown in figure 1.6.2:

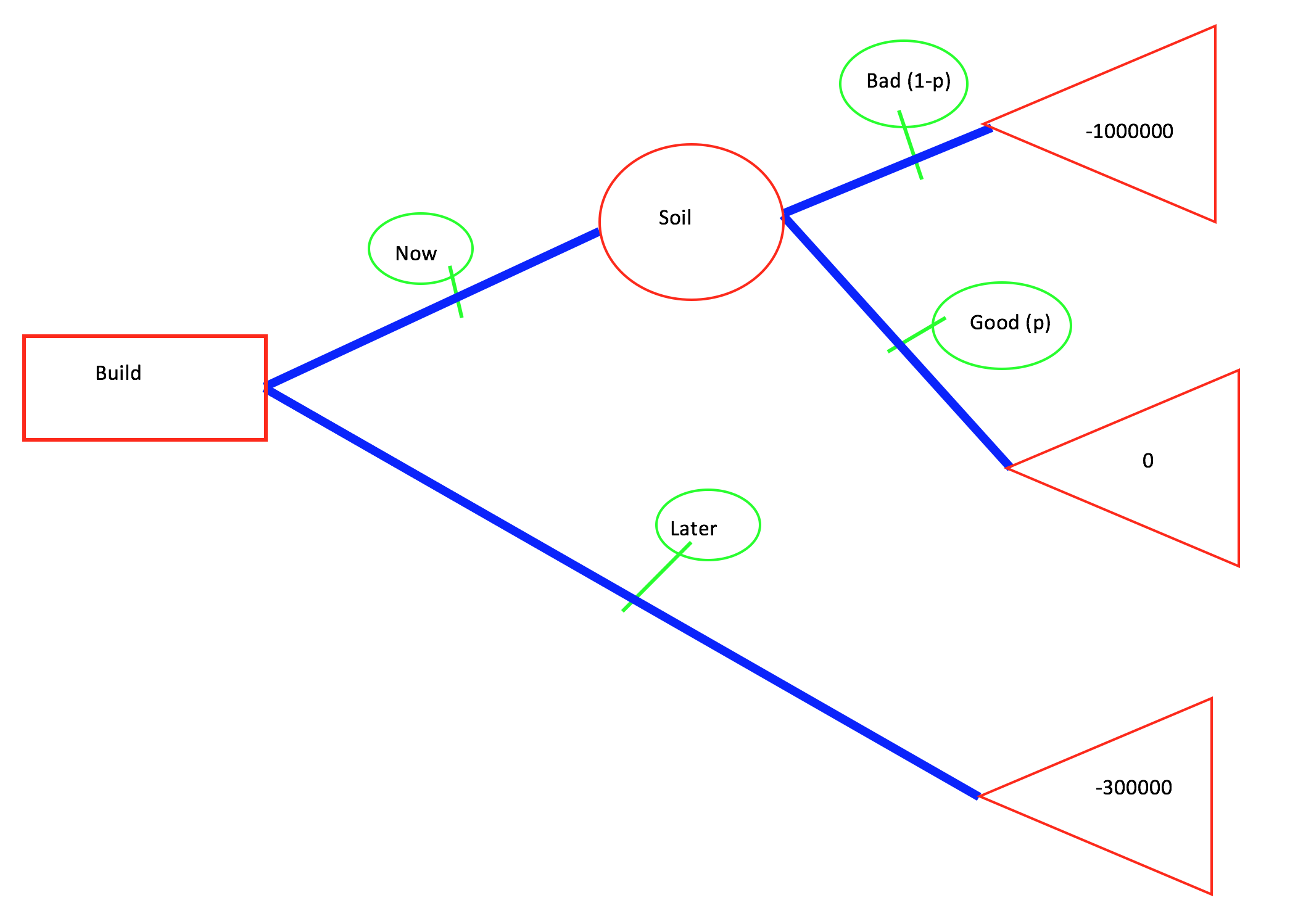


Figure 1.6.2

Showing the color convention in practice for a drawn decision tree; red for nodes, blue for edges, and green indicating the label of an edge.

This color convention was done to reduce the complexity of the problem. If the nodes, edges, and labels of edges are different colors, then using the different channels of a color the algorithm can determine what kind of pixel the particular pixel as apart of.

**Left to Right**: To reduce the complexity of the problem, all trees must have the root node start at the left side of the page with each level of nodes going towards the right of the page. Additionally, each subsequent level cannot loop back and connect to previous levels or start at a location that is further to the left of the parent node location. This is done so the algorithm can tell which node is the root without ambiguity.

**Must touch policy:** All edges must touch nodes, and vice versa. This is done so that the traversing algorithm can only look at neighbouring pixels, greatly simplifying the problem.

1.6.3 How it works

Before the image is converting to an XML file, it is first converted into an intermediate graph structure. The pseudo code to do this is outlined as follows:

1. Convert the image into a bitmap the same size as the image. For each pixel set the bitmap to: 1 if the pixel is an edge, 2 if the pixel is a node, 3 if the pixel is a node label, 4 if node is dark due to overlapping colors. (Pixel value of 4 is useful for mark\_node\_queue\_edges and traverse\_edges\_to\_nodes. Description of why it is useful is out of scope of this description but can be found in the documentation associated with the code).
2. Determine the root for the node by finding the first red pixel in a left sweep search.
3. Recursive step:
   1. mark\_node\_queue\_edges: Mark node pixel as visited and all node pixels adjacent to the original pixel as visited while queueing up all pixel locations of edge pixels adjacent to any of the node pixels visited. Create graph with associated location features. If no edge pixels that are not visited are found, end recursion.
   2. traverse\_edges\_to\_nodes: mark all edge pixels adjacent to the queued up edge pixels as visited. While doing this, queue up all node pixels adjacent to any of those edge pixels and all edge label pixels adjacent to any of the edge pixels. Determine how many unique children there are. For each unique node, recurse back to 3a, and set output as a child of the current graph node.

Once the graph is built, the algorithm goes through each node and edge label and does word recognition using the *Tesseract* library, and then determines the shape of each node using contours. Once the shape and label of each node and the label for each edge is determined, the graph structure is converting into a XML file, which is then returned as the answer. If any of the algorithms fatally fail like an edge not being able to be associated with a label, the algorithm will return a string describing the problem instead of an XML file.

1.6.4 Workflow

The overall workflow would go like this: The instructor from the main page will select the page *Grade Exams*. Once they are at that page, they create the answer for each question that has to be graded from the exam using the online tool to create a decision tree. Once the answers have been established. They will type in the *StudentID* of the student, select the question to be graded, and then scan in the page of their exam that has their solution. The computer vision algorithm will recognize which part of the page the drawn decision tree is on, and then interpret the drawn decision tree into a file that can be read by the grading algorithm. The computer vision interpretation of the given drawn student tree is then displayed to the instructor, and if the interpretation of the drawn student tree is wrong then the instructor is able to edit it until it matches what the student drew. Once the drawn solution is the same as what is one the website, the tree is automatically graded. It does this for each question for a single student and then it goes to the next student.

1.6.5 Limitations

While being able to convert a scanned tree and store it onto our digital tree would be convenient and reduce the change in workflow for both teachers and students, our group has discovered that there are some limitations on what’s possible. Students answers, especially during testing environments, can vary immensely between handing writing, node structure, and organization of the tree itself. As such, certain restrictions must be made on the students when drawing decision trees as described in section 1.6.2. However, it is likely that some students will not adhere to the rules because they are in a stressful situation like an exam.

Even if a student were to adhere to all the rules and wrote very neatly, it is not certain that the Optical Recognition (OCR) software *Tesseract* will correctly identify the labels for the nodes or edges. For example, the software was run on Figure 1.6.2 and the output is shown below:

Parent :: Build

Child 0: Soil, Edge: Now

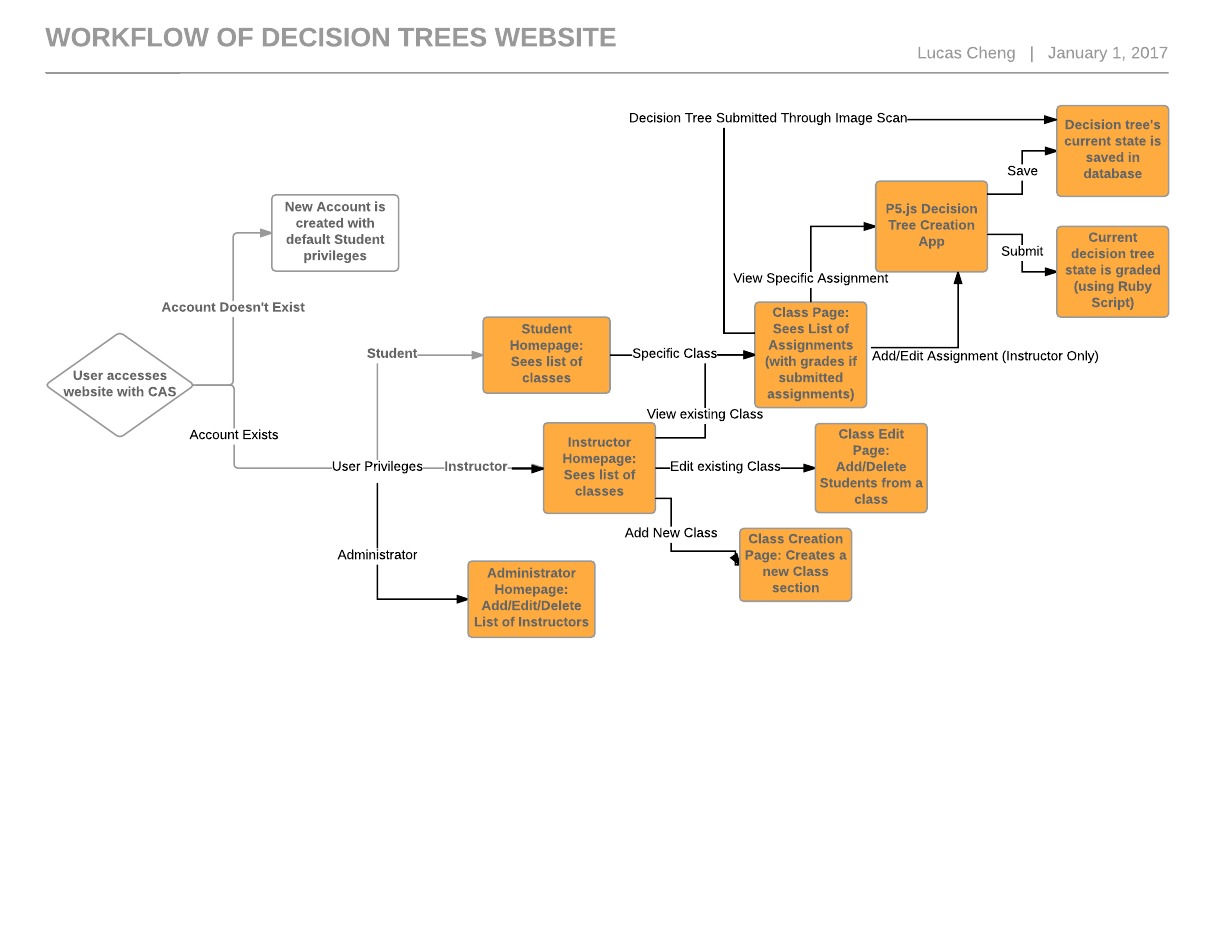
Child 0: -1000000, Edge: Bad (1-p)

Child 1: , Edge: Good (p)

Child 1: -300000, Edge: Later

An indent on the next line indicates the node is a child of the node. The first word after “Parent” or “Child” is the label of the node, and if there is a an “Edge” after the label it indicates the label of the edge going from the parent to the child. Here it correctly identified nearly all of the nodes and all of the edges. The only thing *Tesseract* did not identify was the label for “Child 1” node of the “Soil” node, which was “0”. In this instance, the writing was perfect (printed) and all of the edges and nodes were connected perfectly, and it still did not identify the tree completely perfect. This indicates that it is extremely likely that it would perform badly with hand drawn trees, even if they were neat. More experimentation is needed to make the label recognition software more robust, and it is likely it will not be fully operational by product delivery.

**1.7 Website Structure**

This website uses Ruby on Rails framework with a Puma server and SQLite database. All queries are made in ActiveRecord.

When users first access the website, they will be redirected to The University of Maryland’s Central Authentication System (CAS). The user will enter a password and username. If login is successful, website receives the user’s University of Maryland ID (UID) from CAS. Website checks the database user table for matches on UID :

If not matched: user will be sent to a register form, where they enter their full name, and a user table entry will be created. User will then be directed to the student root page.

Note: If user would like to request instructor status he or she would need to contact the local administrator, or any user with an admin account.

If matched: Booleans in the user table will be checked for instructor and administrator roles. When instructor is true, user will be sent to instructor homepage. Otherwise user will be sent to student homepage.

1.7.1 Student pages

Student home page will have a list of links to the users classes.

Note: Instructor needs to add all students to their classes before they appear on this list. If a student has not yet been added to any classes, this list will be empty.

User will choose from the list of classes and this will take the user to the class homepage. This page will have a list of assignment with due date and grade, if graded. User can then choose an assignment, which will launch a P5 window (described in section 1.4) where they will complete the assignment and submit for grading.

1.7.2 Instructor Pages

Instructor home page will have a list of classes, links to create class, and edit class.

If a class is chosen: User will be taken to the class homepage which has a list of created assignments, and buttons to create assignment, and edit assignment. If buttons are pressed, a P5 window will be launched where the user creates or edits the tree that will be used for comparison in grading. The user can then save or save and submit the tree. Save will convert the tree to an XML format described in section 1.5 and then save to database. Submit will trigger the grading function on the tree and return the grade. If user chooses an assignment that was already created they will be taken to a page with a list of students, a link to their submissions, and their grades if the submission was submitted and graded.

If create class is chosen: User will get to a form. User will fill out class name, and a list of UIDs of students enrolled in the class. An entry to the class table will be created. For each UID, if the UID is absent from the user table, this will create initial and incomplete entries into the database's user table. These entries will be completed when the user logs in for the first time. Also an entry to the student table will be created which will be related to both the user and the class table entry.

If edit class was chosen, user will get to a page with option to add or remove a student by entering the UID and choosing from a list of the Professor’s classes.

1.7.3 Administrators

If the user is an administrator, the class list page will have an administrator tool button. This will take the user to a page with the option to add or delete a user’s professor and admin status by UID. Initial administrator will be Dr. Jeffrey Herrmann. He will be seeded into the database at deployment.

At any point the user can logout by pressing the logout button on the top right.

When a student submits an assignment, P5 converts the tree to JSON or XML. This creates a submission entry in the database where the tree is saved as a string variable. After the entry is created, the tree is sent to the grading function. The functions algorithm compares the submission to the professor’s tree and grades accordingly. When graded, the user will get a pop up with the result.

**1.8 Dependencies:**

1.8.1 Imaging

Below is the software and version numbers for the software that must be installed to run the imaging software as tested:

* Python 3.6.3
* Python Imaging Library (PIL) 2.7.0
* OpenCV 3.3.1
* Tesseract - OCR 3.5.1
* Pytesseract 0.1.7
* Numpy 1.13.3

1.8.2 Grading

* Ruby 2.4.2

1.8.3 Website

* Ruby 2.4.2
* Rails 5.1.4
* SQLite 3.16.2
* Puma 3.7
* Ominauth-CAS 1.1.1

**1.9 Terminology:**

The following terminology shall be used throughout the proposal. The meanings of each phrase is defined below, and shall be consistent throughout the rest of the document.

* **Advanced Tree Comparator:** Compares two decision trees in their canonical forms
* **Answer:** The student’s submitted Decision Tree for the assignment.
* **Backend:** Databases and corresponding API’s used by the system.
* **Basic Tree Comparator:** Compares two decision trees
* **Canonical Solution:** Standard form of representation of the solution for a given problem.
* **CAS:** University of Maryland’s Central Authentication System that takes in a user’s UMD directory ID and password.
* **Decision Tree:** A graphical representation of a series of choices or chance events that show all possible conclusions from the series of events.
* **Edge:** Ordered pair of nodes. Specifies a source and destination node and contains data that results from taking the specified path.
* **Grade:** A number (0 - 100) assigned for the student solution. A higher number equates to a better match.
* **(Grading) Algorithm:** The set of rules defined that compares a student answer to the canonical solution and assigns a grade.
* **Label:** The associated name assigned to a node or edge.
* **Node:** Fundamental unit in the tree that contains information up to that point. Each node has a name and some data associated with it. There are three distinct nodes:
  + Decision Node - symbolized by a square, shows a branch where choices can occur
  + Chance (or Probability) Node - symbolized by a circle, shows a branch where chance events occur through probability
  + End Node - symbolized by a triangle, shows the end result of events through that specific path of choices and chance events
* **Website:** The online tool used by students to create, submit, and receive grading on their assignments involving decision trees.

**2. Intellectual property statement**

The intellectual property (website, grading algorithm, computer vision software) are owned equally by each member of the developer team as a whole. The product is designed for students and instructors who both have access to CAS authentication. The prototyping phase has restricted access to exclusively Dr. Jeffrey Herrmann of the mechanical engineering department at the University of Maryland.

All of the intellectual property of the assignments are owned by the instructor who made them, and any information of the instructor’s respective students is owned by the instructor. All grades automatically processed or given in any way can be changed by the instructor at any time for any reason.

If data is lost on the website due to an unexpected event, the creators of the website are not liable for lost data. The creators of the website are not liable to contact the instructors if information is lost, stolen, or for any reason corrupted. The creators are not required to backup any of the data stored on the server. In addition, the creators are not responsible for the illegal distribution of data in the event of cheating or plagiarism. Instructors are highly encouraged to save their assignments and student grades in another medium website or database.

This software is designed to be able to be distributed and be used by other developers. While we are fully responsible for our own code, we are not responsible for our dependencies that we use (as outlined in section 1.8). If other developers want to use this product on their own server, the original developers must be compensated through an agreement document developed between the original developers and the customers. Once the new developers use the software, they are fully liable for anything that happens, not the original developers. Once the new developers obtain the software, they can change the underlying software if it is agreed upon in the agreement document specified before.

**3. Acceptance test plan**

**3.1 Testing Assumptions**

1. Supported web browsers for purposes of testing and usage are Mozilla Firefox (version 56+), Google Chrome (version 62+), Apple Safari (version 11+), and Microsoft Edge (version 38+). All other browsers and versions are unsupported and used at the user’s own peril.

**3.2 Website Testing**

1. Can the website be installed on a new server?
   1. Desired outcome: Website should have the ability to be installed on a new server by a random person with no prior experience installing a server.
   2. Test:
      1. Create a new VM from the new VM server provided by Purtilo, with no files or installed features on it
      2. Get a random UMD student who has no affiliation with CMSC435.
      3. Ask student to install the website on the new VM using the instructions provided by our team, using versions specified by in our instructions.
      4. If the student is able to without any intervention by the team, then test is passed. If not, the test is not passed.
2. Can the user log in?
   1. Desired outcome: When logging in, does the user get redirected to the UMD CAS login page? After the user logs in, do they get redirected back to the home screen of their account?
   2. Test administration account
      1. Add one of our own group’s ID to have Administration privileges
      2. Test to see said person has power to add others as Instructors
   3. Test student (Default user role):
      1. Get random engineering student from Professor Herrmann’s class
      2. Ask them to login to the website through the help of CAS authentication
      3. Check if they get redirected back to their home-page (check the icon that shows their ID)
      4. If they do, test is passed
   4. Test Instructor (requires an administrator account to allow Herrmann as an Instructor):
      1. Ask Herrmann to login to the website
      2. Check if they get redirected back to their homepage (check the icon that shows their ID)
      3. If they do, test is passed.
3. Can an instructor create a class?
   1. Desired outcome: Instructor can create a class on the website and add student IDs to the website so that they can join
   2. Test:
      1. Instructor creates a new class, through a link on in their profile page.
      2. Instructor adds the UIDs of other students in Decision Trees.
      3. Other students in decision trees log in.
      4. If all students who were authorized are able to log on and see the created class, test is passed. If not, test is not passed.
4. Can an instructor create an assignment?
   1. Desired outcome: Instructor is able to create an assignment for a class and the students in the class are able to see the assignment.
   2. Test (assumes an administrator has a class created):
      1. Administrator creates an assignment for the specified class. The assignment can be any decision tree that can be made on the website.
      2. Students in the class log in and see if they can view the assignment
      3. If the students can see the assignment, test is passed.
5. Can a student answer a question?
   1. Desired outcome: A student fills out an answer on an assignment. The answer gets saved and can be viewed by the administrator and the student.
   2. Test (assumes there is an assignment already created by the administrator):
      1. Student from the class opens up the assignment.
      2. Student completes assignment using the online tools (Section 1.4) to create a tree.
      3. Student saves answers.
      4. Student goes out of the assignment and checks to see if their answer was saved as the answer for the assignment.
      5. Administrator logs on and checks if the student(s) completed the assignment and the answer submitted by the student.
      6. If part (iv) and (v) both passed, test was passed.
6. Can a student/instructor see their grade?
   1. Desired outcome: Students are able to see the grades given to them by the grading algorithm. Administrator is able to see the grades of all the students in the class.
   2. Test (assumes administrator created an assignment and some students completed it):
      1. Student logs into the website. Goes to their grade page. If the student did not complete the assignment, their grade should be 0. If they did complete the assignment, their grade should be the one given by the grading algorithm.
      2. Administrator logs into website. Goes to their grade page. All grades associated with their students should be there. The grades must be the same as the grade shown to the students.
      3. If part (i) and (ii) are both passed, test is passed.

In addition to these functional tests, we plan on evaluating our product after walk through. This plan involves taking actual students of Dr. Herrmann’s class and letting them use our deployed website (with some sort of compensation or incentive for the testing students) and receiving feedback through a survey on the ease of use of this product compared to creating trees by hand, and if they value the faster response time that this product would give.

**3.3 Grading Testing**

1. Does the basic tree comparator accurately compare two trees?
   1. Desired outcome: Basic Tree Comparator accurately identifies two equivalent trees to be equal and two different trees to be different.
   2. Test: (Assumes two different decision trees)
      1. Plug in both decision trees to the basic tree comparator.
      2. Compare the two trees to check for inequality
      3. If part (ii) passes for multiple, randomly selected-sized trees, test is passed.
   3. Test: (Assumes two equal decision trees)
      1. Plug in both decision trees to the basic tree comparator.
      2. Compare the two trees to check for equality
      3. If part (ii) passes for multiple, randomly selected-sized trees, test is passed.
2. Does the advanced tree comparator break down every tree into its accurate canonical form?
   1. Desired outcome: Advanced tree comparator collapses every tree into its accurate canonical form (two logically equivalent trees will collapse into the same tree)
   2. Test: (Assumes two canonically different decision trees)
      1. Plug in both decision trees to the advanced tree comparator to obtain their canonical forms
      2. Compare the two trees to check for inequality
      3. If part (ii) passes for multiple, randomly selected-sized trees, test is passed.
3. Does the collapsion of canonically equivalent trees lead to a unique tree?
   1. Test: (Assumes two canonically equivalent decision trees with different expanded forms)
      1. Plug in both decision trees to the advanced tree comparator to obtain their canonical forms.
      2. Compare the two trees to check for equality
      3. If part (ii) passes for multiple, randomly selected-sized trees, test is passed.
4. Is the grading algorithm accurate in grading?
   1. Desired outcome: Graded decision trees are within the predetermined accuracy range of within Herrmann’s expectations
   2. Test: (Assumes both a student decision tree and a teacher’s correct answer key decision tree exists, and both trees have been previously unseen by the teacher and algorithm)
      1. Plug in both decision trees to the grading algorithm
      2. Have a teacher grade the student’s decision tree by hand
      3. Compare the percentage given by teacher to that by the grading algorithm. Subtract from each other and determine if it’s within the predetermined range, with a bias towards lower grades
      4. If part (iii) passes for multiple, randomly selected-sized trees, test is passed.
5. Is the grading algorithm faster than by hand?
   1. Desired outcome: Graded decision trees are graded faster than by hand
   2. Test: (Assumes both a student decision tree and a teacher’s correct answer key decision tree exists, and both trees have been previously unseen by the teacher and algorithm)
      1. Plug in both decision trees to the grading algorithm
      2. Have a teacher grade the student’s decision tree by hand
      3. Compare the time taken for teacher to grade vs. the grading algorithm
      4. If grading algorithm takes less time for multiple, randomly selected-sized trees, test is passed
6. Can the grading algorithm identify which nodes are incorrect?
   1. Desired outcome: The grading algorithm correctly identifies which nodes from a student submitted answers are different than the instructor’s answer key
   2. Test: (Assumes both a student decision tree and a teacher’s correct answer key decision tree exists, and both trees have been previously unseen by the teacher and algorithm)
      1. Plug in both decision trees to the grading algorithm
      2. Manually check and highlight where the student’s tree differs from the accepted answer tree
      3. If grading algorithm finds less than 70% of differences found manually between the trees, the test fails
      4. Repeat for multiple, randomly selected-sized trees, and if it succeeds for all of them, the test passes
7. Can the grading algorithm identify synonyms, truncations and misspellings?
   1. Desired outcome: The grading algorithm can mark all variations of a correct label as still correct, without taking any points off.
   2. Test: (Assumes an “altered” tree exists with synonyms, truncations, and misspellings of correct labels)
      1. Plug in tree with altered labels and tree with correctly spelled labels into grading algorithm
      2. Compare grades between two trees
      3. Test partially passes if the grades are the same between the two
      4. Plug in tree with intentionally incorrect labels and a tree with correctly spelled labels into grading algorithm
      5. Compare grades between trees
      6. Test passes if incorrect labels are given a lower grade - fails if they are still given the same grade

Immediately after the walk through, we envision the professor conducting several ungraded in-class exercises with different classes involving several decision trees that the student will create and submit to receive feedback on. During the process, students will be able to leave feedback on how streamlined they felt the overall process was, which we envision to be positive as we designed the system with simplicity and efficiency in mind. After the in-class exercise is complete, the students would be surveyed with basic short-response questions where they will be able to comment on how they would feel adopting an automated grading system would be and how accurate they felt the grading would be. At this point the workflow and grading algorithm should be well defined so we hope to only receive requests for improvement on small details rather than any large design flaws, and we will view success here as not having any major complaints on our implementation.

**3.4 Imaging Testing**

Warning: There is a risk that the technology being tested by the tests listed below is not going to be fully mature by the time tests are done, and thus might not pass by the time of product delivery.

1. Can the imaging recognize a perfectly drawn solution? (figure 1.6.2)
   1. Desired outcome: Imaging software is able to recognize an ideally drawn solution.
   2. Test: (Assumes imaging software is run via command line and not integrated into the website, all necessary software is installed, python3 is python 3.5)
      1. On command line, type ‘python3 img2xml.py /path/to/figure1\_6\_2.png’, where ‘/path/to/figure1\_6\_2’ is the path to figure 1.6.2 as a png file
      2. Answer should be identical to Figure 3.4.1. If the output is identical, test passed. If not, test failed.

<xml>

<decision n = "Build" e = "null" p = "null">

<final e = "Now" v = "-300000" p = "null"/>

<chance n = "Soil" e = "Later" p = "null">

<final e = "Good" v = "0" p ="p"/>

<final e = "Bad" v = "-1000000" p = "1 - p"/>

</chance>

</decision>

</xml>

Figure 3.4.1

Correct XML file structure representing Figure 1.6.2

1. Can a colored over black and white perfect drawing be read? (figure 1.1)
   1. Desired outcome: Imaging software is able to recognize an ideally drawn solution that has been color annotated by an instructor.
   2. Test: (Assumes imaging software is run via command line and not integrated into the website, all necessary software is installed, python3 is python 3.6.3, markers used to trace over are sharpie pens in the respective colors)
      1. Print out Figure 1.1 on white paper.
      2. Color annotate print out:
         1. Trace over node borders in red, edges between nodes in blue, and around edge labels in green. An example is figure 1.6.2.
      3. Scan in file using a color scanner.
      4. On command line, type ‘python3 img2xml.py /path/to/figure1\_1\_annotated’, where ‘/path/to/figure1\_1\_annotated’ is the file path to figure 1.1 annotated as a png file.
      5. Answer should be identical to Figure 3.4.1. If the output is identical, test passed. If not, test failed.
2. Can a colored over black and white hand drawn tree be read? (figure 1.1)
   1. Desired outcome: Imaging software is able to recognize a hand drawn solution that has been color annotated by an instructor.
   2. Test: (Assumes imaging software is run via command line and not integrated into the website, all necessary software is installed, python3 is python 3.6.3, markers used to trace over are sharpie pens in the respective colors)
      1. Have a random student from the decision tree group draw figure 1.1 within the guidelines specified in section 1.6.2.
      2. Color annotate the drawing:
         1. Trace over node borders in red, edges between nodes in blue, and around edge labels in green. An example is figure 1.6.2.
      3. Scan in file using a color scanner.
      4. On command line, type ‘python3 img2xml.py /path/to/hand\_annotated’, where ‘/path/to/hand\_annotated’ is the file path to scanned picture as a png file.
      5. Answer should be identical to Figure 3.4.1. If the output is identical, test passed. If not, test failed.
3. Does the workflow outlined in section 1.6.4 work?
   1. Desired outcome: Workflow proposed for the instructor works and streamlines the process of grading.
   2. Test: (Assumes all necessary software is installed, python3 is python 3.6.3, markers used to trace over are sharpie pens in the respective colors, the decision tree we are grading is figure 1.1)
      1. Administrator creates a class with a single student with the studentID “jj100”.
      2. Go to main page. Click “Grade Exams”.
      3. Provides the solution using the online tool. Use figure 1.6.2
      4. Go through steps 3.b.i to 3.b.iii
      5. Test passes if the drawn tree gets full credit. If not, test failed.

**4. Cost estimate**

**4.1 Function Point Analysis**

**ILF (Internal Logical Files)**:

Looking throughout our own database, we determined that we’ll have a total of 20-50 data types stored in around 2-5 relational databases. Fields include strings for storing user information, as well as what role they have, XML (which is stored as another string field) for the storage of decision trees, floats for grades, and timestamps for submission times. Overall we have 5 relational databases as of this time - students, instructor, users, projects, and class. As such we have an overall cost of an average complexity table, giving a base value of 10 points. After taking the complexity of the overall table, we determined that most of the relations between the database tables are still simple, giving us a total of 36 points.

**Total: 36 points.**

**ELF (External Logical Files**)

For the most part, external interfaces calculates anything that references data that isn’t maintained by us, and is from an outside source - like CAS or Wordnet, both of which we are using. As of now, we anticipate using around 4 large external libraries and/or code repositories within our project: p5 javascript library for displaying tables, rwordnet as a Ruby gem to help get synonyms for our grading algorithm, CAS to authenticate users, and openCV3 eventually to try and image process simple trees. However, even though we are using 4 different libraries, we aren’t receiving and using these libraries to process many different data types, meaning we still have a low complexity, giving us a base value of 7 Points. While we have a base low complexity and process a low variety of data types, the relationship of these libraries is still unfortunately quite complex in how they are an integral part of each of our relative sections. The website can’t function if the user can’t see the tree (p5.js) or login (CAS). The grading algorithm would only be able to match one word, or rely on a wordbank without rwordnet. And without openCV3, we aren’t able to post-process simple written trees to detect nodes. As such, we evaluate a total of 40 points.

**Total: 40 points**

**EI (External Input**)

External Input applies to all input that users will be inputting to the website - meaning log-in pages, answers to questions, creating classes, instructor permissions all count. While the majority of the important input will be the decision tables created and stored within our application, the total count of user input far exceeds this one type. Overall, we have around 14 different inputs - including CAS log-in, class, instructor and assignment creation, along with question prompts and decision table creation. Taking into account the types of databases these inputs are connected to, we have a total of 6 base points. While most of these inputs are very simple, the one decision tree input is decidedly very complex, giving the input complexity to be average. This gives us a total of 24 points.

**Total: 24 Points**

**EO (External Output**)

This subgroup takes into account any data that we send out to the user, as long as the data is internally processed and displayed. As such, there’s quite a lot of different outputs besides merely our grade that we give to each decision tree for each student. We additionally output feedback on the graded tree, such as highlighting nodes and labels where the graded decision tree different from the correct decision tree. If we get image processing done, an additional output of the tree in digital form would also be needed. Thus, taking everything into account, there will be about 4 major outputs needed with high complexity - giving us a total of 28 points.

**Total: 28 Points**

**EQ (External Queries)**

External Queries applies to all outputs that are not processed, and simply gives information back to the user. Examples include our questions that the teacher poses to the students, UID information, class information, and other various pieces of information transferred between instructor and student roles. Taking into account everything that we need to display, there are about 4 different main outputs that would be taken into account without processing - displaying student profiles, assignments (the text to decision tree questions), instructor profiles, and the home page. However, these outputs are all from differing database tables, giving us a base value of 4 function points. The complexity of these things are quite low, since referencing and displaying everything is quite trivial. This gives us a total of 12 function points.

**Total: 12 Points**

**Grand Total: 140 Points**

With this taken into account, we further plan to divide the overall points into their individual subgroups - where we can better learn how much each section plans to take, and if one particular needs more or less people in general.

**4.2 Website Cost Estimate**

This is the largest part of our project, and thus contains the majority of the functional points. We can assume that almost all internal logical files, external inputs, and external queries apply to the website - giving us a subtotal of 72 points. For external logical files, about half of the cost is from website, (CAS and p5 libraries), which add 20 more points. This gives a total of 92 points.

**Estimated Sub-Group Total: 92 Points**

For our website part, there are currently three people who are responsible for designing, developing, and testing the frontend and backend of all website services. Taking into account that this subsection costs about 92 Points, we decided to estimate on the longer side (as our team needs to learn quite a lot of new material) as working at 0.5 points/hour. In addition, we’ll have a group meeting once a week about 2 hours. With only 3 weeks remaining, we have a total of 202 man-hours dedicated to this project. This translates to about 68 hours per week - and with 3 people in this group, averages to around 23 hours per person per week.

**Estimated Total Man-Hours: 202 Hours**

The website can generally be divided into two main parts - the frontend and the backend. The frontend involves mainly the UI that the user will interact with. This means any HTML, CSS, and Javascript work. Our most important component will be the decision tree creation portion that the user sees, and for the decision tree structure to be correctly transferred and stored to the back end. Breaking down our functional point analysis even further, we can determine that most External Inputs fall under the frontend category - as does External Queries. In addition, one library (p5.js) also falls under this category from External Logical Files. This gives a total of 46 Points, or 92 man-hours for the frontend. While our function point analysis assumes no work has been done so far, thankfully we have made quite some progress in this part - we have a functional frontend where “clicking” a predetermined area produces a decision-tree node that is correctly stored in p5.js. The user can additionally add children nodes, with connected edges, and select a particular node. Adding that this work took around 20 man-hours, we estimate that we have around 72 man-hours left. As the group total has 3 people, we have one person (Greg) primarily working on this frontend, with a second person (Liu) working between frontend and backend. Dividing the total cost of 72 man-hours - Greg will take 48 hours, and Liu will take 24 hours on this project.

The backend deals with data manipulation, storage and transfer and typically consists of three parts: a server, an application, and a database. This most directly translate into writing code for both models and controllers after receiving input from the frontend. In particular, the most important part of this will be the javascript object that the backend will receive from p5.js and the ability to convert this to a standardized XML format to be stored and able to be used by our grading and imaging components. In general, the priority of work for our backend in model and controller creation will be Users > Instructors> Students> Assignments> Trees, with the addendum of creating an XML parser to store decision trees first. As the frontend took 46 function points, conveniently the backend also just happens to take around 46 function points - meaning another 92 man-hours for the backend. And as with the frontend we have also made some progress in creating the schema (database tables) for the website, as well as a basic user model. While the progress here isn’t as far as the frontend, the work done still takes around 8 hours off, leaving us with 84 man-hours left on the back-end. Just like the frontend, we have two people working on the backend with one person as the primary person on backend (Yoni) and one person (Liu) as the person between frontend and backend. Dividing the 84 man-hours cost of the backend, we have Yoni taking 56 hours and Liu taking 28 hours.

Weekly Breakdown

Week 1 (Includes already finished work): Website setup

For our first week, we anticipate setting up the database, creating CAS functionality, and setting up the different user roles that go along with it. Basic web page functionality should be present, as users can login and see their profile, as well as any assignments or classes they may be in. Questions for the decision trees, which is just text, can be created, but the creation of decision trees themselves on the website will not start until the next week. As such, we anticipate this will take around 60 man-hours to set everything up out of the 202 total man hours predicted, since a majority of the time will also be learning how to set up everything in Ruby (for Liu and Greg who doesn’t know Ruby as well) - meaning less time will be spent on actual coding.

Week 2: Website Development

By our second week, we should have most of the website set up - with all user roles defined. Upon logging in through CAS, the website should be able to automatically direct users to either a student (default) or instructor account. In addition, the functionality for users to create decision trees should be at least in some finished version in the frontend, even if there isn’t any connection with the backend yet. Instructor roles should be able to create classes and add students to them. Since a majority of the development is occurring here, there will be a marked increase in spend work hours needed - increasing to around 90 man-hours.

Week 3: Website Refinement and Testing:

By this time, we should have a working website with full functionality - with full ability to create decision trees, and ability to store the tree on the backend. This week will mainly focus on making everything more user friendly, and accessible. As such, increased attention will be paid on the front-end as we work on our website’s aesthetics and in reducing unnecessary links. Since the main work on the website should have been completed by Week 2, we anticipate this week will take the remaining 52 man-hours of the predicted 202 hour total to develop the website.

Week 4: User Testing Phase:

This is when we have already deployed and have finished the walkthrough after the hard deadline. We anticipate still taking feedback from users to improve our website to facilitate use and hopefully help the transition to a more widespread use of our application to reduce workload on UMD instructors. During this time, we also anticipate the use of surveys on Herrmann’s students to see exactly if our product was able to solve the problem we were tasked to solve. With the feedback, we anticipate using around 40 more man-hours in distributing surveys, as well as adjusting anything on the website that users deemed as a defect.

**4.3 Grading Cost Estimate**

While a smaller portion of the project is technically focused on the grading subgroup, the impact of this module is significantly more important as it’s a crucial component in our effort to solve the problem of grading graphical assignments. As such, it is absolutely paramount that this grading algorithm work correctly and quickly. Breaking down our main functional points, we find that parts of External Logical Output as well as External Output almost all fall under our grading algorithm. Adding up the cost of related libraries (rwordnet), as well as output in grades and highlighted decision tree nodes gives us a total of 31 Functional Points.

**Estimated Sub-Group Total: 31 Functional Points**

This is significantly less than our Website sub-group, and with three people on the team, we ideally should finish ahead of our walkthrough date so that Herrmann will be able to see the results of our grading algorithm and offer critiques or improvements to make the grading algorithm more precise. In addition, our Grading subgroup feels that 31 Functional Points is still a lower estimate, due to the complexity of possible decision trees and that it doesn’t fall under a typical data structure. As such, we aimed for a higher estimation of 40 Functional Points, due to the added complexity (since there was not much a gradient for complexity). This means that to complete this sub-group, our team will need to put in a total of 80 man-hours. With three people in this current subgroup, this amounts to around 27 hours per week, or 9 hours per person per week. As such this group is much more relaxed in amount of work, it is much more likely that some people of this group will help the website group (especially once the initial grading algorithm is done). Again, this cost estimate assumes that no work has been done when we have already developed the basic node-comparison portion of the grading algorithm in ruby (along with basic class structure within ruby) and also installed and successfully used rwordnet. As such, our compensated estimate for the grading sub-group is a total of 70 man-hours, with 10 hours about completed. The breakdown of hours can be relatively equal - with each person (Ameen, Lucas and Hugo) taking around a total of 24 man-hours. However, as this workload is noticeably less than those working on the website, it is highly likely that some of the website workload will be transferred to the grading sub-team, such as translating p5.js objects into XML to distribute the workload more evenly.

**Estimated Total Man-Hours: 70**

Weekly Breakdown

Week 1 (Includes already finished work): Tree Setup Phase

In this phase we plan on creating the “skeleton” of the program. We will setup the environment and configure it to be able to write our functionality. After doing so, we plan on building a “Basic XML Tree Comparator.” This functionality will allow XML trees to be built, parsed, and compared. At this stage, we will be able to verify the equality of two trees assuming all nodes and edges are equal. While two engineers will be building this, we will also begin researching and designing the grading algorithm which will be implemented in the coming weeks. The total estimated cost for this phase will be 20 hours out of the 70 total hours.

Week 2: Grading Phase

The grading phase is the one most fundamental phases for the project. The beginning of the week consists of implementing the “Advanced XML Tree comparator.” This is an extension of the basic comparator that will allow trees of different structures to be compared using a collapsing algorithm. This will be a precursor to the grading algorithm, as it will collapse every tree to its unique form and allow for a more accurate result when graded. While implementing this comparator, we will also be finishing the design of the grading algorithm. In the second half of the week we will have finished the design for the grading algorithm and implement it. While the grading algorithm is being implemented, we will also be writing black box tests for the algorithm. These will check to make sure that the a given tree receives the expected grade from our algorithm. The total cost estimate for this phase will be 50 hours.

Week 3: Deployment Phase

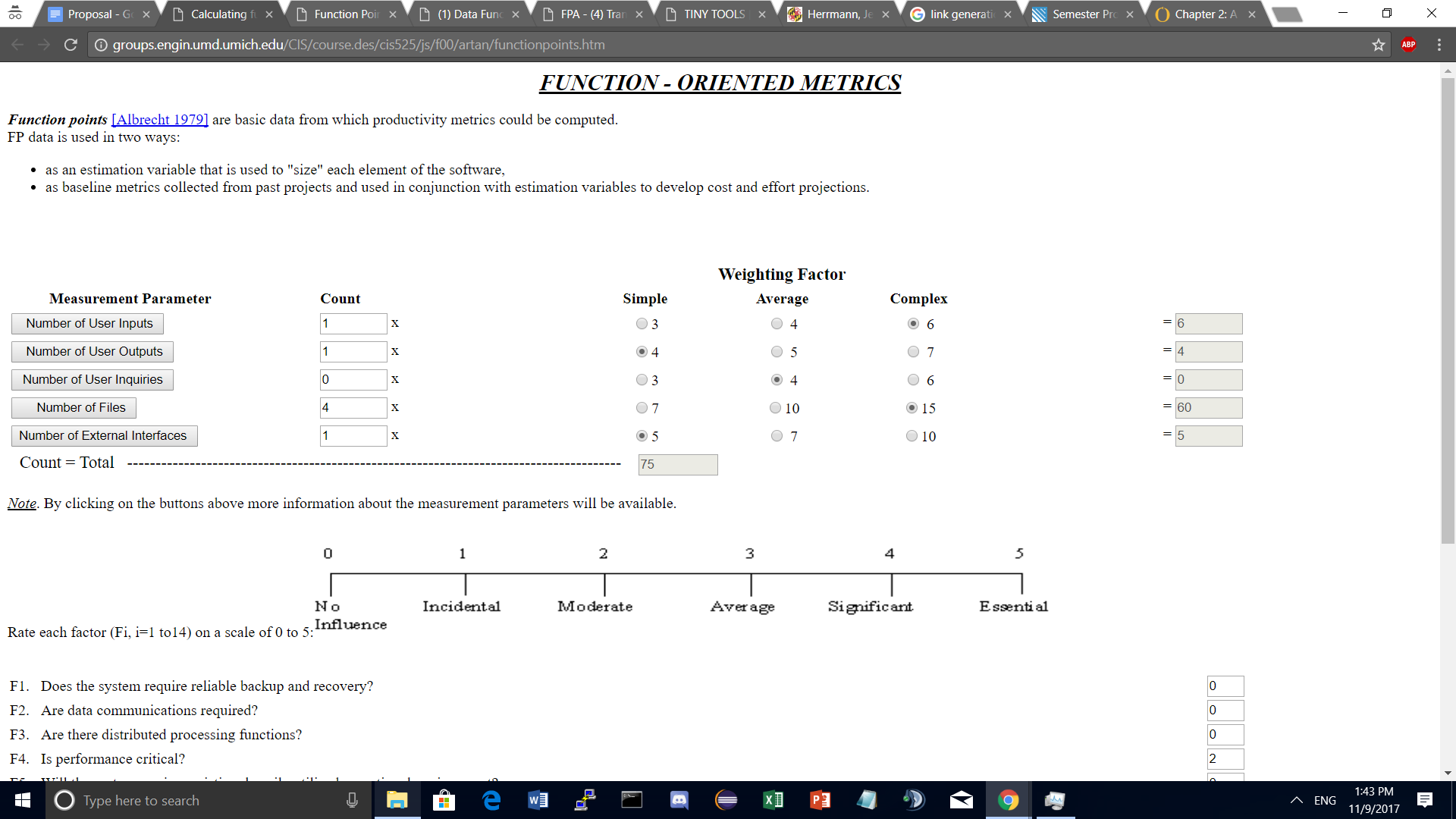
By this week, we will have the general functionality set up. Using some of the tests from Week 2, as well as some white box tests written from our Website subgroup, we plan on optimizing the grading algorithm with Herrmann to make sure it runs in the most efficient and accurate manner. Starting after the team meeting on Wednesday, we will have all 3 members of the team focus on deploying the back-end functionality. The total cost estimate for this phase will be 30 hours. This exceeds our 70 hour estimate in that this after developing the product, and is our response any response given to the finished grading algorithm.

Week 4: User Testing Phase

This week was purposely left relatively “light.”, not because we believe that there will be a light workload, but to allow extra time for any delays and problems we may face. We allocated 3 days to fix any bugs or add optimizations that we find, but we are aware that this may take much longer. We plan on adapting based on the amount of delays/problems we experience, and will allocate time for the rest of the week accordingly. The current allocated time for this phase will be 15 hours, since this is after we have showcased and deployed our product (hopefully without any bugs) but can and will be adjusted accordingly.

**4.4 Imaging Cost Estimate**

The hardest to estimate, and quantify in terms of what we expect to do, is our imaging subgroup. As of this time we currently have one person, David, working on the imaging aspect who is attempting to convert scans of physical copies of decision trees into digital ones. From our Functional Point Analysis at least, including External Logical Output and External Output, we only have 17 Functional Points assigned to this particular subgroup. However, as this amount is not even close to truly estimating the cost of the actual scope of our imaging, we decided to break from our overall project estimation, and redo a calculation based on what David was doing. The following adjusted values were put into a function-point calculated (Figure 1.3):



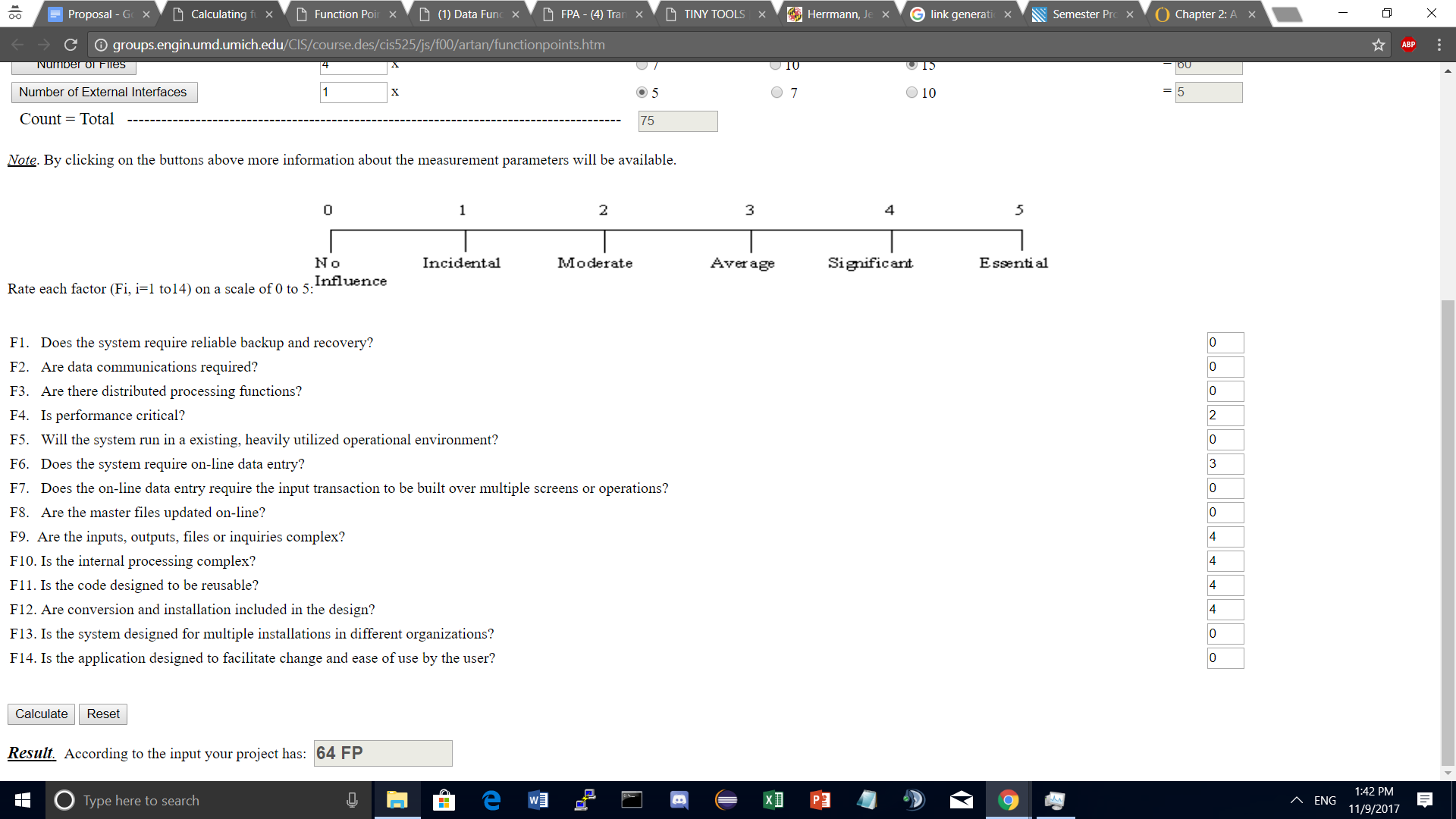


Figure 1.3

As seen, the new estimated total for our imaging portion is around 64 Function Points - which converts to around 128 man-hours. Around 20-30 hours has already been completed by David and his interaction with Professor Purtilo, who has been helping him with various machine learning techniques. This still leaves around 100 man-hours to complete this task, which is incredibly unrealistic for one person. As such, this is due to the very ambiguous and non-critical nature of this task. While it is preferable to finish this task, there are likely to be many limitations on the picture that will be scanned and converted digitally - meaning that this subgroup unfortunately won’t probably see much use even if it is finished. Even so, we fully realize and anticipate helping David as a group once other mission-critical sections of the project are finished (such as Grading).

**4.5 Maintenance fee estimate**

After our project becomes live and populated with users, there is a very slight possibility of our website will becoming overloaded with the amount of traffic and data. As such, we would need to find someone who would update the data, transfer the data to whatever grading service the teacher uses, and who would make sure there’s enough space for new data (assignments). We will need to make such a person an administrator who can directly access to the database to help the instructor load and remove students, as well as modify data if an instructor or student needs to change it (such as grades). Said person would probably do the maintenance work for probably one hour a week. As such, this role would probably fall under the responsibility of a hired T.A. - who should be able to manage the website after gaining permission from the instructor.

Another idea is to use AWS(Amazon website services) to keep safe our data in a safe and stable environment insteading of using local server. The advantages of using cloud computing and storage are quite clear in that it’s flexible, cost-effective, scalable and has high-performance. The disadvantages would be mainly the price, as the price for a database is about 0.29 per hour multiplied by 0.052 per GB stored. We can also consider other company’s services but the price are mainly relatively same.

**5. Timeline**

* Sunday, October 29th
  + Website
    - Finish the CAS and Devise, set up database
    - Test the CAS and Devise database verification
* Monday, November 6th
  + Grading
    - Comparing trees absolutely (canonical perfect match?)
      * Either 100% or 0%
    - Parse into accurate tree representation
    - Collapsing the tree into its canonical form
  + Website
    - Build generally each web page structure, and routes
    - Test page structure by feeding independent data
* Monday, November 13th
  + Grading
    - Be able to compare sections of the tree for partial credit
    - Obtain a similarity score for both the collapsed and uncollapsed trees when compared to original
  + Website
    - Finish website sketch decision tree part and enable loading data from trees.
    - Finish frontend with j5 to create decision trees
* Wednesday, November 15th
  + Grading
    - Start comparison of labels in addition to node structure
    - Testing of rwordnet synonym capability
  + Website
    - Start of parser of XML to p5.js object, and vice versa
    - Finalization of creation of user roles (Instructor/Student)
* Friday, November 17th
  + Grading
    - Be able to highlight nodes where students got off
    - Start of testing synonyms with labels in grading trees
  + Website
    - Integrate with the back-end algorithm part, so that they can receive information from database and sending information to database.
    - Enable to display student grade on our web page
    - Finalize XML parsers
* Monday, November 20th
  + Grading
    - Finalize testing of synonym functionality
    - Start parser from XML to Ruby object to read-in from website
  + Website
    - Addition of decision tree creation to all relevant pages
    - Finalization of CAS login to direct to correct user role (Instructor/Student)
* Wednesday, November 22th
  + Grading
    - Finalize XML Parser to Ruby Object
    - Assist with Website subgroup in XML parser to p5.js if needed
  + Website
    - Finalization of Student and Instructor profile pages and Assignment pages
    - Filling class page’s relationship together with students, instructors and assignments
* Friday, November 24th
  + Grading
    - Testing of whole algorithm against Herrmann’s grading
    - Assist in integration with website
  + Website
    - Integrate with Imaging subgroup, so that we also can load information from images.
    - Testing all the feature and finding bugs and weakness, having users outside our group access the website
* Wednesday, November 29th
  + Website and Grading
    - Last minute touches on Project.
    - Whole product interaction testing, from all groups

**6. Project advertisement**

*Decision trees* team offers an online system that will facilitate the construction of decision tree assignments. It will allow the professor to post questions and correct answers to each corresponding question in decision tree assignments. Students will be able to view each assignment as well as each question on each assignment, and they will be able to construct, save, or submit a decision tree using an online interface. In addition, the professor will be capable of scanning in images of hand drawn decision trees for exams. Assignment submissions and scanned exam trees will be stored, constructively graded, and displayed along side their grade for the professor and the corresponding student to view.

Our plan for advertisement integration is to select a simple, minimalist open source CSS template from either <https://templated.co/> or W3Schools for integration into into the Professor’s web page that will present a quick overview of our final product. The final presentation will have overview pages describing the functionality of our product, of how the automation of grading decision trees will be beneficial to those who use decision trees in instructional classes, and a page with our p5.js decision tree creation tool. Within the overview page there will be navigation links to a setup walkthrough, and an information page. The setup walkthrough link will provide instructions on how a user can host their own version of our decision tree formation and grading website, along with needed dependencies and their relevant documentation and download links. The information page will outline the purpose and reason for creating our product, along with repeating several of the key features such online tree submission and automated grading.