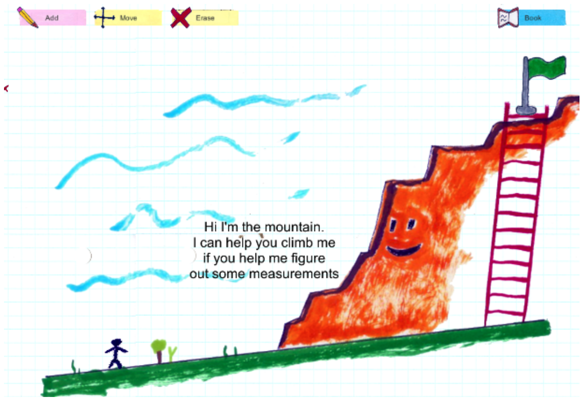
Triangle Peak: An Educational Trigonometry Game

Maximilian Levine
Department of Computer Science
University of North Carolina at Asheville
Email: mlevine@unca.edu

***Abstract*—Triangle Peak is an educational game that turns learning trigonometry into a fun puzzle platformer. It is inspired by practical applications of trigonometry, and transforms what would be dry textbook problems into something that feels real. Gamification is a practice that often involves attaching RPG-inspired leveling systems onto otherwise unrelated items. But Triangle Peak uses a different and genuine educational approach, with its mechanics directly mirroring the aspect of reality it corresponds to, and through uses of trigonometry in the real world, places the player into practical usage of the concepts. The completed product of Triangle Peak has great potential to be used in education, and to allow people of all ages to get a better grasp on trigonometry concepts in a fun way.**

I. Introduction



Hi I'm the mountain. I can help you climb me if you help me figure out some measurements

Fig. 1: Climb the mountain.

Triangle Peak was created when it was noticed how game-like certain problems look in textbooks (Figure 2), and a system was developed that would allow these and other similar problems to be played in the form of a video game.

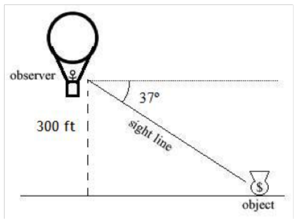


Fig. 2: Game-like textbook problem.

The player must traverse a world and find or calculate the measurements of varying things—width of a chasm, height of an air balloon, depth of a well, etc.—to progress, using their mathematical toolset to find the answers just out of reach. It melds platforming with trigonometry problems by turning walls and the limitations of gravity into the unknown lengths/angles found in regular trigonometry problems. The player uses a drafting book that lets them draw and connect lines on the screen, and a measuring tape and protractor that respectively lets them measure the lengths and angles of these lines—provided their player character can physically reach them. The player also collects various trigonometry equations: the "SOH CAH TOA" equations, the Pythagorean theorem, the sum of angles rule, similar triangles, law of sines/cosines, and so on (Figure 3). They can plug in their measurements to these equations and receive outputs for the unknowns in the problems.

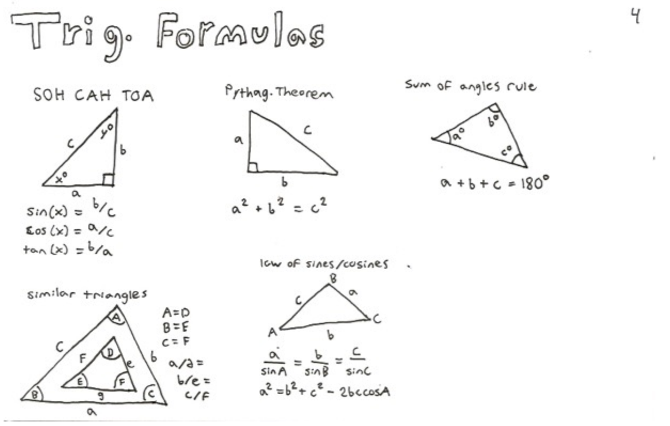


Fig. 3: Some trig equations.

So here are the core systems: the player character can move by walking and jumping. The user can draw and connect lines on the screen, which snap to each other, forming triangular shapes. When the player character touches two sides of a line, or angle between lines, the length or angle (respectively) appears as a draggable block. A sheet of equations can be accessed any time, and contains trigonometric equations. The previously outputted values can be dragged and plugged into the equation (similar to how equation blocks work in Scratch); when only one unknown remains, its value is outputted.

The resulting system produces a game that emulates many trigonometry problems in the real world, and by placing the player into these scenarios, the player is sure to learn and understand trigonometry concepts on a deeper and more intuitive level. There is also the opportunity for an additional research study: seeing how effective the game is at teaching trig, compared with traditional textbook methods.

II. Background

A. Similar software

I first wanted to look into Geometer's Sketchpad (GSP), which is an educational program used to do geometry and trigonometry. It was useful to look at since I would be implementing a subset of its features with the line/angle drawing/measuring systems.

Another program to compare is Scratch, a programming system for kids developed by MIT. In Scratch, equations and variables are like Lego blocks that can be snapped together (Figure 4). This is similar to how equations will work in Triangle Peak. Again, it will be useful to see how these are implemented.

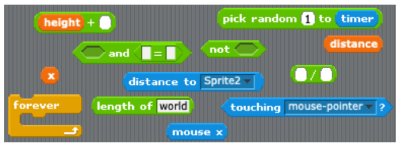


Fig. 4: Expression blocks in Scratch

Foldit is a game about folding proteins that benefits real science (Figure 5). This game melds game with science and education: every aspect of its mechanics parallel what it is teaching about.

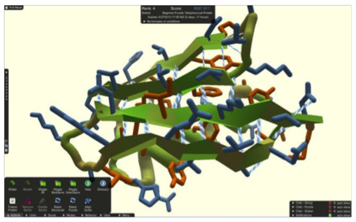


Fig. 5: Foldit protein folding game

World of Goo (Figure 6) is a physics platformer game about building structures out of triangles, so it has some things in common with this project. Something might be able to be gleaned from the freedom with which players are able to explore the world through triangles.

Fig. 6: Goo tower in World of Goo

B. Literature review

Almeqdadi examined Geometer's Sketchpad, the educational program used to do trigonometry. In this study, one group of students used GSP, while a control group simply used the traditional textbook. Pre-tests and post-tests were performed, and the result was that, compared with the control group, GSP significantly improved the students' performance. (Almeqdadi)

In another study on GSP, Nordin did an extensive study taking place in 34 mathematics secondary schools. A digital module was developed using Macromedia Authorware to be used in conjunction with Geometer's Sketchpad, allowing for a more focused use of the software. The results show that the methods met educational requirements, and the authors suggest looking further into using GSP in mathematics teaching. (Nordin)

In an article on educational games, Valerie claims that perhaps even a game not as explicitly educational as the GSP modules can promote learning. World of Goo is a physics platformer game about building structures out of triangles, so it has some things in common with Triangle Peak. Something might be able to be gleaned from the freedom with which players are able to explore the world through triangles. And the section of this book, "Does Playing World of Goo Facilitate Learning?" discusses and makes the claim that World of Goo itself is already educational, despite not being explicitly so. As Henry Ford says, "Failure is the opportunity to begin again, more intelligently." (Valerie)

In an article titled, "Math Games: An Alternative (Approach) to Teaching Math," Eliens goes on to discuss using new media strategies and games to promote learning. This article is all about using new media to teach math, which they believe will get students over the common, initial hurdle of fearing math. New media allows "constructive explorations in a wide variety of mathematical problems." The article discusses games as instruments to drill concepts, but also as ways to think big-picture and identify strategies for problems, something that World of Goo is a great example of, and that Triangle Peak is built around.

Jean Justice looked into barriers for using games and simulations for education, and surveys a broad overview of the topic of simulation games (which Triangle Peak is), as well as the barriers that educators perceive from adopting simulation games, such as an aversion to the word games, and ways to avoid those barriers. (Jean Justice)

In an accessible paper, Liu discusses simulation games, which have a high potential for users to enter a "flow" state. The paper tries to answer the question of how simulation games improve problem solving by running an experiment on 117 students in a simulation game designed to target computational problem solving. (Liu)

Cruickshank in the academic paper "Classroom Games and Simulations" covers the effects of games and simulations being used educationally. Of interest is its distinction between games and simulations, the main difference being that games have a specific goal. This is similar to comparing GSP with Triangle

Peak, since the latter will essentially be a more focused version of the former, and with a goal. (Cruickshank)

In “Trigonometry: Comparing Ratio and Unit Circle Methods,” Kendal discusses the teaching of trigonometry, and compares two common approaches of doing so: the old way, and the new way along with the advent of “new mathematics.” This is useful to see the comparison, and the most effective, proven ways of learning trigonometry. (Kendal)

“Trigonometry” by Gelfand is a popular book all about trigonometry, seemingly the standard on the subject. This is useful to learn/relearn some trigonometry concepts, such as trigonometry identities, that could be used in the game. It starts from the basics and works its way up. (Gelfand)

Buchberger in “Computer Algebra” surveys and explains methods of computer algebra systems, i.e. equation solvers, which could be needed in Triangle Peak. These systems are highly complex. For this reason, it seems like it will be best to either use an existing library or to hard-code code each equation in the game manually. The benefit of the former, though, would be flexibility and the potential for users to enter their own equations. Ultimately, we went with the hard-coded approach. (Buchberger)

“The Best Game Engines for Making Your Own 2D Indie Game” by Famularo was useful in determining which engine to use for Triangle Peak; ultimately we went with Unity. This article compares different game engines, specifically for 2d projects. (Famularo)

In article “The Guide to Implementing 2D Platformers” on HigherOrder Fun, Monteiro goes into an in-depth tutorial on how to build 2D platformers. It is fairly comprehensive and also discusses slopes, stairs, and ladders, which are included in Triangle Peak. This is a valuable resource for building the platforming aspect of the game. (Monteiro)

III. Project Description

This game required the programming and implementation of some complex systems, the use of a game engine and related software, as well as the challenge of deploying and publishing a game—all of the problems that creating an original, commercial game entails.

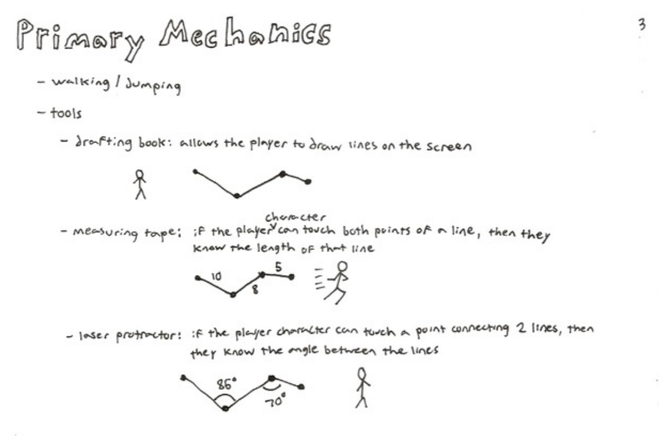

Primary Mechanics
3
- walking / Jumping
- tools
- drafting book: allows the player to draw lines on the screen
- measuring tape: if the player character can touch both points of a line, then they know the length of that line
10
8
5
- laser protractor: if the player character can touch a point connecting 2 lines, then they know the angle between the lines
86°
70°

Fig. 7: Protractor and measuring tape.

A. Requirements/Specifications

One challenge in developing Triangle Peak are the complex and nontrivial software systems that must be built. These systems are custom and specific, such that I can’t simply follow a tutorial to implement them, but must engineer new solutions myself.

1) Line snapping: The most challenging of these features is the line drawing mechanic. The user must be able to draw and connect lines. Points should be able to connect to other points, lines, or intersections. Additionally, points should ”snap” so that the user can form precise constructions without having to ”eye-ball” it. These snapping areas are defined and allow for snapping to form specific lengths and angles. Whenever a user drags a point, it should snap to these snapping lines/points if near enough.

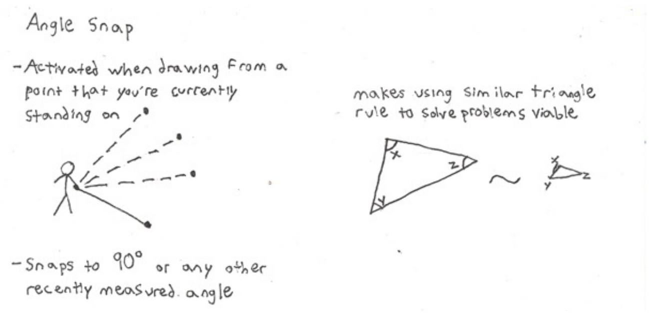

Angle Snap
- Activated when drawing from a point that you're currently standing on
makes using similar triangle rule to solve problems viable
x
y
z
- Snaps to 90° or any other recently measured angle

Fig. 8: Angle snapping.

2) Math problems: When the player character activates a line or node, the correct lengths and angles must be calculated and appear in the correct places. There were also many other cases where challenging, yet specific and modular, math calculations needed to be coded, and these required referring to researching external math formulas and algorithms.

3) Book of equations: Another issue entirely is the book of equations. The user can drag values into the various trig equations, and the equations must then output the correct value once there is only one unknown value.

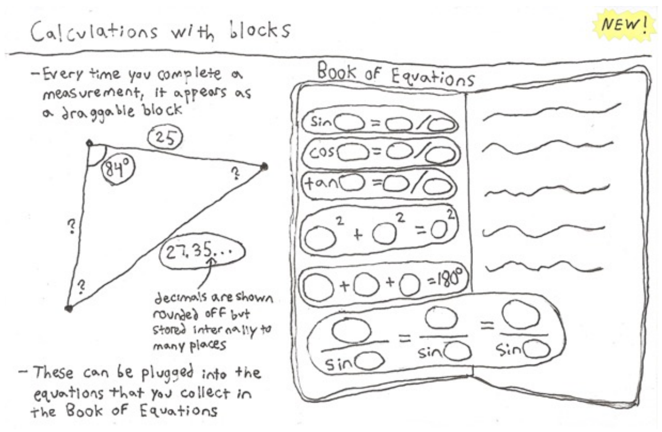

Calculations with blocks
NEW!
- Every time you complete a measurement, it appears as a draggable block
25
84°
?
?
?
27.35...
decimals are shown rounded off but stored internally to many places
- These can be plugged into the equations that you collect in the Book of Equations
Book of Equations
sin = /
cos = /
tan = /
2 + 2 = 2
+ + = 180°
sin = sin = sin

Fig. 9: Book of equations.

4) Platforming: Standard 2d platforming will also be needed, with slopes and ladders.

5) Aesthetics: A paper aesthetic guides the art direction, as if the game takes place entirely in someone's notebook.

B. Design

1) Line snapping: The line drawing system took most of the development time. It was made more challenging due to round-off error, making it impossible to rely on specific coordinates as being accurate. In order to make this system, the use of a graph data structure - with nodes and edges connecting them - was utilized. The storage utilized was a list of node objects with integer ids, each of which had a list of all their adjacent nodes. Each of these node objects also had a reference to its corresponding game object, which has coordinates and appears on the screen. A separate list of edge line objects (which connect any two nodes) is also maintained, such that for any operation, only one edge line object between any two points exists at a time.

Originally, in the minimum viable product (MVP) prototype, users would draw a single line segment, and on releasing the mouse, this line segment was incorporated into the main graph structure, appropriately connecting the new segment to any nearby nodes. However, in order for all of the snapping features to work, it was necessary to change this (so that duplicate code for segments and the main structure wouldn't have to be made). So, when the user draws a line, now both nodes are immediately placed into the structure, while the player's mouse is dragging the other end of the line. All consolidated into one structure, the advanced snapping features were made simpler.

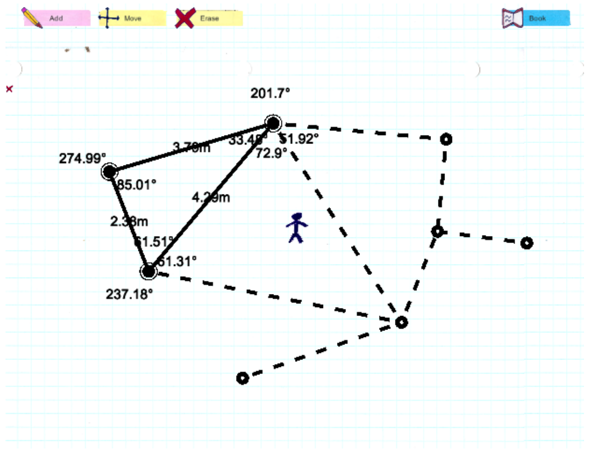


Fig. 10: Line construction

The design for the snapping system is theoretically simple (although it took 2,000 lines of code). While a node is dragged, look for something to snap to. If there is, receive the coordinates to snap to as well as a unique snapping string. At this point, a backup of nodes, edges, and their activations are recorded, for undoing later on. After this backup is made, alterations to the graph structure appropriate for the snap type are made. There are four types of snap: real node snap,

	how snapping affects graph	how it is calculated	unique/extra info stored
real node	both nodes (current moving and this one) are combined, i.e. their edges are added together	in graph nodes	
imaginary point	for any edges found to intersect this point, disconnect the nodes on either side, then reconnect those nodes to the current moving point	any intersections between lines and/or imaginary lines	all edges found to intersect at a given point are stored in a dictionary
line	the nodes on either side of the edge are disconnected, and each are reconnected to the current moving node	in graph edges	a virtual line is stored that the node must snap to
imaginary line	no effect on graph	these consist of lines forming angles, lines forming a 0 degree angle (self snap), as well as circles (for distance snap)	

imaginary point snap, line snap, imaginary line snap. And for each of these, respectively, there are several ways in which the graph structure is modified: both nodes are merged (their outgoing edges are combined into one node), any edges that have been found to intersect the imaginary point have their points disconnected and then reconnected to the current node, the points joining the edge are disconnected and reconnected to the current node, and lastly no changes are made for imaginary lines. Each of the imaginary types must be calculated prior to this, and are formed from intersections, complement angles, and circles formed for getting specific lengths. Once successfully snapped, the moving node continues to query to see if it should still be snapping. Thus it continually receives the snap id string as well as updated coordinates (the line snapping states can be the same snap id but with new coordinates). If the snap id is different (the player has moused away from where they were previously holding the node), then the node must unsnap, reversing changes made to the graph structure by referring to the backup. This effectively allows one to drag a node anywhere and for it to move freely while still snapping to potential nearby points. Once the mouse is released, the node is permanently in its place in the graph, and the backup data is discarded. Refer to the very general pseudocode for how all of this comes together.

```
NodeObject:

  members:
  string snapId
  bool isSnapping
  Vector2 position
  Vector2 snappingPoint

  while being mouse dragged:

    if not isSnapping:

      (isSnapping, snapId, snappingPoint) <- Snap.
          snap(position)

    if isSnapping:

      position = snappingPoint;

      (newIsSnapping, newSnapId, newSnappingPoint)
          <- Snap.snap(mousePosition)

      if snappingPoint != newSnappingPoint:

        (isSnapping, snapId, snappingPoint) <- NULL

Snap:

  snap(position):
    calculateImaginaries()
    (isSnapping, snapId, snappingPoint) <-
```

```
      getNearest(position)
    backupGraph()
    incorporateChanges(snapId)
    return (isSnapping, snapId, snappingPoint)

  members:
  list of imaginary lines and circles
  dictionary where imaginary point —> list of
      intersecting edges

  calculateImaginaries:
    createComplementAngleLines() //imaginary
    createLengthCircles() //imaginary
    foreach lines (and circles):
      foreach lines (and circles):
        if intersection:
          save imaginary point
          also add the intersecting lines to
              dictionary with point as key
            (unless imaginary, then don't add)

  getNearest(position):
    search real graph and all imaginaries stored in
        snap for
    nearest within snapping distance (points
        prioritized over lines)

  incorporateChanges(snapId):
    case real node:
      combine current node with found node
    case imaginary point snap:
      foreach line intersecting:
        remove edges from either point
        add edges to current point
    case line:
      remove edges from either point
      add edges to current point
    case imaginary line:
      nothing
```

2) Math problems: For calculating lengths and angles of activated nodes, these required some standard formulas. In addition, calculating intersections of lines, circles, and rays, required standard formulas. Calculating the distances between and among points, lines, rays, and circles, too, required standard formulas. Creating the right complement angle ray to a line used a rotation with a 2d quaternion. These are all standard and can be found easily in a math book or StackOverflow.

In order to figure out which angles to display, I had to sort the points emanating from a point in clockwise order. I also needed clockwise order to figure out of there was room for a complement angle to form; if a complement angle will actually extend past an already existing line, then that complement angle is not created. The following line of comparison can be used to see if point a is before or after point b in clockwise order around a center point (and then used in a sort function):

$$atan2(a.x - \text{center}.x, a.y - \text{center}.y)$$

$$< atan2(b.x - \text{center}.x, b.y - \text{center}.y)$$

3) Book of equations: For equations, a more flexible option was considered, which could allow the user to input their own equations, would be to use some kind of equation solving program. Finding an equation solver library could speed this up, make it easier to add new equations, and, best of all, perhaps allow users to enter their own equations. This idea was foregone for a hard-coded, manual approach. Using the Pythagorean theorem, $a^2+b^2=c^2$, as an example, three cases would need to be written, depending on which unknown value needs to be outputted. See the pseudo-code block for a segment showing the Pythagorean theorem.

4) Platforming: Standard 2d platforming was implemented using Unity's Collider, Effector, and Rigidbody components, and off-the-shelf code. As this is not a platforming focused game, careful configuring wasn't needed. However, functional diagonal ladders were a unique aspect, and were adapted from a tutorial on straight ladders.

5) Aesthetics: A combination of masks, shaders, materials, as well as drawn and scanned hand-drawn paper textures and illustrations were used to produce the paper aesthetic.

```
function getUnknownInEquation(equationType, unknown,
    known[]) {
  switch(equationType) {
    case pythagoreanTheorem:
      switch(unknown) {
        case 0:
          return sqrt(known[0] - known[1]);
        case 1:
          return sqrt(known[0] - known[1]);
        case 2:
          return sqrt(known[0] + known[1]);
      }
    ...
  }
}
```

C. Required Resources

1) Software: One important question concerned the game engine to use. Options included Unity, Godot, and GameMaker. Unity is very popular but a bit bulky with features that wouldn't be needed and has some annoyances when working in 2D. Godot is a little more lightweight option, as well as free to use commercially. GameMaker is great for this type of 2D game, but uses GML as its language, has less useful facilities as other languages, and is sometimes looked down upon for being unprofessional/for nonprogrammers. Ultimately, I decided to go with Unity, since I have more recent experience using Unity. And because of the large scope of Unity, this will potentially allow for additional, notably/primarily visual, enhancements down the line. Unity can easily export to many platforms.

Another question we had to determine was which platform to develop for. At first, I wanted to target multiple platforms, starting with tablet and PC. Instead, it was decided it would be better to focus entirely on one platform, at least starting out, which will be desktop PC, in addition to HTML/web as this platform is relatively easy to port between.

D. Testing/Validation

The part of the program with code paths complex enough to afford testing is the line drawing system. (However, equations could also be tested by trying all possible equations and unknown value combinations.) One line of testing is a stress test: randomly drawing a bunch of lines while moving the player character erratically. Due to some inefficient solutions, the game can run a bit slower with many lines onscreen at once. However, there is no reason for a user to actually do this. The game did use to crash when doing this, due to entering into an infinite loop; this was in the main snapping step, where the NodeObject would try to snap again immediately after unsnapping. But this has been identified and resolved.

Another way to test is to make sure that all snapping categories work correctly. There are points, edge lines, angle snap lines, self snap lines, circles, and directly connected circles (the snap functionality shown earlier in pseudocode only has to be familiar with four broader categories). And all of these intersect with one another and produce imaginary points, with some slight rule variations on whether imaginary points should form at all from circles. To test this, imaginary points should be deemed as functioning as expected for all combinations.

E. User Verification/Feedback

A study can potentially be performed to see if the game is more effective at teaching trigonometry than traditional methods, such as through a textbook. To perform the study, we can form two groups: one who plays the game, and the other who reads a textbook description. A literature review is conducted to find the usual methods of teaching trig, for the textbook case. The groups then take the same trigonometry test. We hypothesize that those who play the game will perform significantly better on the test. These results will be applicable to the applications of games, simulations, and new media for education.

Initial play tests showed that users were confused about how to play/proceed, and thus didn't enjoy the game. This appeared to be over confusion that one can drag the values onto the equations while the book is open, and making this more easily communicated would improve usability greatly, as it appeared to be the main stumbling block. This will be worked on to be improved.

IV. Results & Functionality

The resulting program is a fairly robust geometry sketch tool. It is fun to mess with. One can draw and connect points and lines in an intuitive way. The lines and points can snap and reconnect in a way that makes sense. One can form specific lengths and angles. And these work in conjunction with the player character. When the player stands on a node, the correct lengths and angles appear in the right places. If a node is moved and the player moves, these measurements are updated dynamically, and the movement and connectivity of the nodes is compatible and built around the idea that the player can activate them at any time, even in mid-snap state. Moving nodes or recombining them angle/length activates/deactivates as expected.

In conjunction with the equations sheet, dragged values have an arrow pointing to where they formed from, which makes a construction look like an illustration of a math problem. The book of equations can be used to drag the generated values and calculate answers to the problems correctly. The player character can move around in a physical world. After constructing lines, measuring them, and performing calculations on the results with equations, the player has the correct value, and this is verified by the game, which allows them to proceed to the next level.

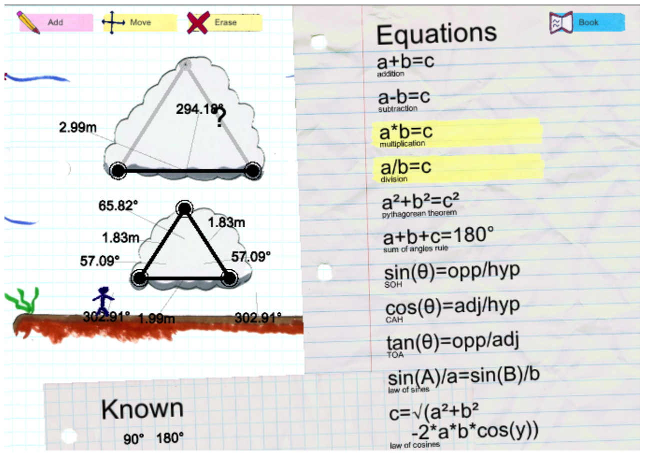


Fig. 11: All the elements of a level.

V. Discussion & Reflection

Figuring out how to implement this was a mental journey, one that I wasn't sure that I would be able to make, but I am proud that I was able to. There were many long moments of bug fixing or being at a loss of figuring out how to incorporate a needed feature, as well as times of scribbling notes to myself when I found a solution. I may have in fact gotten to deep into the geometry system, or into a rabbit hole as Dr. Whitley said. This was in part due to perfectionism, as I wanted the lines to be completely done before moving on. The lines are still not completely done. There is more potential there currently, but all that it amounts to in the levels are simple constructions of triangles, where the equations system does most of the work. The changed discussed in the Future Work section could remedy this and to allow for this to be expanded into a more fully featured geometry tool.

VI. Future Work

Our solution was challenged by round-off error, which meant that relying on specific coordinates and easily transferring them to points and other objects were unreliable. A future approach to reconcile this would be to create an abstract space to mediate, where points that are similar enough are simply combined. But this may not be completely necessary, as our results have shown that the system in place does work as is.

There are several issues with the snapping system to be polished over. One is that, when a line is extending, it won't deactivate the other nodes, if there are multiple possible extension points, and it also won't deactivate correctly as long as it is snapping to something. Another issue is that sometimes snapping simply feels buggy. These issues can be resolved by adding more information along with the snap id: Which point produced the imaginary point, and does it need to continue to be activated to snap to? Does the player character need to be in snapping distance to this new point for it to snap? And what is the exact imaginary point, to be used in the dictionary to find all intersected edges?

Also, more fully introducing a circle structure which is visualized for the player could improve the systems. Currently, the circles used for length snapping are implicit and not displayed, but making this more overt to the user could improve the functioning of the system, as currently there is confusion on whether a circle should even generate an imaginary point, which this would resolve.

An angle snap circle emanating from the current moving node is currently not generated, which means it's not possible to form an exact angle when moving the node your character is standing on. The reason for this, is that this snap shape would only be a circle for forming 90 degree angles; for other angles, it would actually be a superimposed set of ellipses (which would need to be rotated arbitrarily as well), which is why this feature is nontrivial but could be added in the future.

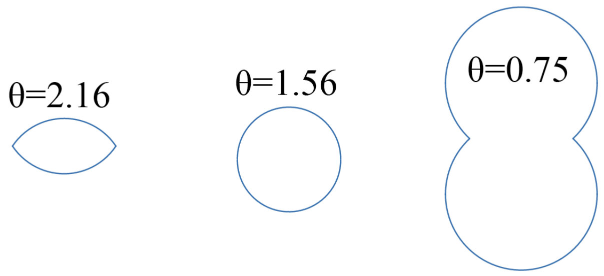


Fig. 12: Shape formed showing where a middle point would have to be to produce a given angle between two points

I would like to license the song ”Particle Man” by They Might Be Giants to use in this game, which is thematically relevant, as the game takes place in an absurd but intellectually curious world. A more expansive world, with more levels and more character dialogue (inspired by the song), needs to be added.

Finally, there are numerous stretch goals that can be added to the project, given enough time, which would involve random generation to create new content, as well as an editor and a server to upload and download custom levels from. This would involve utilizing databases. The randomly generated challenge would involve a new mechanic, in which the player can make physical triangles appear, once all angles on them have been measured, as solid objects that can be climbed on. The physics can easily be implemented in Unity. The goal of the mode is to ascend a randomly generated tower structure. The user generated mode would allow users to place down platforms into an editor, in addition to a length value that needs to be found to win the level; these would then be uploaded to a database for others to play.

I am planning to continue working on this project for the foreseeable future. My next immediate steps are to focus on some of the visual polish: character animations and more paper effects. Then I want to improve the new player experience, so

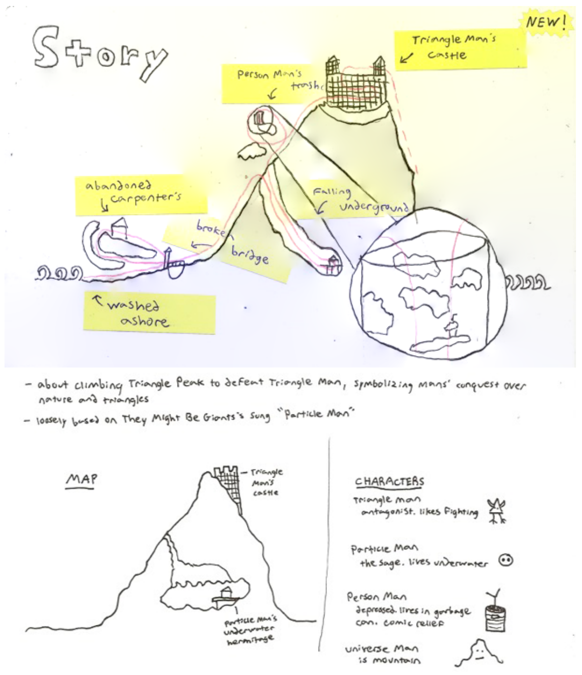

Story
NEW!
Triangle Man's castle
Person Man's trash
abandoned carpenter's
Falling underground
broken bridge
washed ashore
- about climbing Triangle Peak to defeat Triangle Man, symbolizing mans' conquest over nature and triangles
- loosely based on They Might Be Giants's song "Particle Man"
MAP
Triangle Man's castle
particle Man's underwater hermitage
CHARACTERS
Triangle man antagonist. likes fighting
Particle Man the sage. lives underwater
Person Man depressed. lives in garbage can. comic relief
universe Man is mountain

Fig. 13: Idea for more expansive world, characters, and story

that people know they can drag the values onto equations, to get some better feedback. From there, it is back into the geometry system to implement and fix the changes listed above (it's almost there!). And finally, fleshing out the game and expanding its world and story to be an effective teaching tool would be my ultimate goal.

References

Almeqdadi, Farouq. "The Effect of Using the Geometer's Sketchpad (GSP) on Jordanian Students' Understanding of Geometrical Concepts." Proceedings of the International Conference on Technology in Mathematics Education, July 2000, files.eric.ed.gov/fulltext/ED477317.pdf.

Compton, Kate, and Michael Mateas. "Procedural Level Design for Platform Games." Association for the Advancement of Artificial Intelligence, 2006, www.aaai.org/Papers/AIIDE/2006/AIIDE06-022.pdf.

Cruickshank, D.R., and Ross Telfer. "Classroom Games and Simulations." Theory Into Practice, vol. 19, no. 1, 1980, pp. 75–80., doi:10.1080/00405848009542875.

Eliens, Anton, and Zsofia Ruttkay. "Math Games: An Alternative (Approach) to Teaching Math." GAMEON, 2009, research.utwente.nl/files/5381162/paper-math.pdf.

Famularo, Jessica. "The Best Game Engines for Making Your Own 2D Indie Game." Pcgamer, PC Gamer, 28 Sept. 2017, www.pcgamer.com/the-best-2d-game-engines/.

Gelfand, IM. Trigonometry. Birkhauser, 2001. The book can be accessed at https://users.auth.gr/ siskakis/GelfandSaul-Trigonometry.pdf.

Jean Justice, Lenora, and Albert D. Ritzhaupt. "Identifying the Barriers to Games and Simulations in Education." Journal of Educational Technology Systems, vol. 44, no. 1, 2015, pp. 86–125., doi:10.1177/0047239515588161.

Kendal, Margaret, and Kaye Stacey. “Trigonometry: Comparing Ratio and Unit Circle Methods.” Education. Proceedings of the 19th Annual, 1996, citeseerx.ist.psu.edu/viewdoc/download doi=10.1.1.408.6413&rep=rep1&type=pdf.

Liu, Chen-Chung, et al. “The Effect of Simulation Games on the Learning of Computational Problem Solving.” Computers & Education, vol. 57, no. 3, 2011, pp. 1907–1918., doi:10.1016/j.compedu.2011.04.002.

Monteiro, Rodrigo. “Higher-Order Fun.” HigherOrder Fun RSS, 20 May 2012, higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/.

Nordin, Norazah, et al. “Pedagogical Usability of the Geometer’s Sketchpad (GSP) Digital Module in the Mathematics Teaching.” The Turkish Online Journal of Educational Technology, vol. 9, no. 4, Oct. 2010, files.eric.ed.gov/fulltext/EJ908077.pdf.

Shute, Valerie J, and Yoon Jeon Kim. Design Research on Learning and Thinking in Educational Settings Enhancing Intellectual Growth and Functioning, Routledge, 2011, pp. 243–263.

Image References

Foldit. (n.d.). Retrieved October 02, 2019, from https://fold.it/ Scratch. (n.d.). Retrieved October 02, 2019, from https://scratch.mit.edu/ Tbaisd. (n.d.). Retrieved October 02, 2019, from https://www.tbaisd.org/ World Of Goo From 2d Boy. (n.d.). October 02, 2019, 2021, from https://2dboy.com/