Designing an Effective Math Educational Video Game

by

Allison Knox

Submitted to the Department of Math and Computer Science
School of Natural Sciences
in partial fulfillment of the requirements
for the degree of Bachelor of Arts

Purchase College
State University of New York

May 2021

Sponsor: Lee Tusman
Second Reader: Knarik Tunyan

Contents

Abstract

This research considers how philosophies of education apply to educational video game (EVG) design. As educational environments have drastically shifted into virtual spaces during the COVID-19 pandemic, effective virtual educational tools are essential to the learning environment. In pursuit of learning by doing, I design an educational video game in light of the principles of game design explored in my research. In this game, the user is presented with conic sections graphed on a Cartesian plane, such that the user is expected to construct each conic sections' corresponding equations based on its graphical representations. The graphical representations of the conic sections are designed such that they appear as a cohesive image. Thus, the objective of the game is such that the user learns the mathematical relationship between graphical representations of conic sections and their equations. As we navigate the process of designing an EVG, we critique its efficacy in light of the design principles and educational ideologies.

Chapter 1. Introduction

An educational video game, or EVG, is a video game that imparts some educational value onto the user. EVGs fall under the edutainment genre, which is a descriptor for media that incorporates education through entertainment. It is, of course, an ambitious endeavor to set out to create a successful EVG. By considering both "Playability Guidelines for Educational Video Games," and Ian Bogost's *Persuasive Games: The Expressive Power of Videogames*, we will parse out and discuss what truly makes an EVG successful. First, let us consider what makes a video game educational.

As Bogost argues in *Persuasive Games*, discerning whether a video game is educational is a subjective classification. Bogost steps back to first discuss two foundational educational philosophies-- behaviorist and constructivist-- before discussing how they shape interpretations of educational video games. These educational philosophies serve to "guide and structure educational practice," granting us lenses through which to analyze an EVG's educational value (Bogost 235).

First, we will consider the behaviorist educational philosophy. Behaviorism grounds itself in the belief that "psychology is a 'natural science' based on empirical observation," and thus its core education philosophy asserts that "learning [...] takes place via repetition and reinforcement" (Bogost 234, 233). When applied to video games, Bogost argues that behaviorists would seek to construct a microcosm environment that is modeled like a classroom, in which desired behavior is rewarded and reinforced through video game mechanics. In other words, behaviorists would create an EVG that models an educational environment and imparts knowledge to its player through repetition and rewards. Bogost poses *Microsoft Flight Simulator* as an example of a game that a behaviorist would deem successful in its educational objective. *Microsoft Flight Simulator* successfully "transfers its subject matter to the player" by creating a realistic simulation in which the user pilots an

aircraft (Bogost 237). While this educational philosophy may be appealing in its simplicity, Bogost acknowledges there are "objections to behaviorism abound," since this staunchly scientific philosophy "leave[s] no room for human subjectivity" (Bogost 233-234). This is to say that behaviorism is often critiqued for its simplicity, that it fails to consider that not every person learns best through "repetition and reinforcement." Furthermore, analyzing the educational value across other video games can lead to oversimplified analyses. Consider a first-person shooter game through a behaviorist lens. Bogost points out that a behaviorist would assert that this type of game "positively reinforces" violence as "appropriate behavior" (Bogost 236). Ultimately, this interpretation proves that "behaviorist approaches to games foreclose" what Bogost dubs "the simulation gap" (Bogost 238). The simulation gap is the "breach between the game's procedural representation for a topic and the player's interpretation of it" (Bogost 238). A behaviorists' staunch commitment to their educational ideology blinds them to the "cultural nuance and the subjectivity of representation," which excludes the individuality of a player's interpretation and experience with the game (Bogost 238). Indeed, a player's experience with an EVG is paramount when considering its efficacy. This is echoed in "Playability Guidelines for Educational Video Games," which establishes player experience as a hallmark of an EVG's success. Certainly, while behaviorism may be a tempting and simplistic lens through which to construct and analyze one's video game, maintaining a strict behaviorist perspective can overshadow important aspects of a game, like player experience. Having examined behaviorism, let us then turn to constructivism as a fellow contending educational philosophy.

Constructivism asserts that "the learner 'constructs' knowledge individually," such "that learning is inseparable from the learner's interaction with the environment" (Bogost 234). In short, constructivists center the subjectivity of one's experience, because people learn best "by doing" (Bogost 234). A classic example of a constructivist learning

environment is the conventional “kindergarten” classroom, which relies on “play, materials, and activities as a means to encourage creativity and [...] fulfillment” (Bogost 235). Constructivism may seem more appealing to the modern learner, but it too has its shortcomings. Due to its individualist approach to teaching, some social psychologists-- like James K. Doyle-- argue that “changes in ‘thought, behavior, or organizational performance’ are limited to anecdote and bias, with little [...] scientific basis” (Bogost 234). This is to say that constructivism as an educational philosophy puts too much emphasis on the subjective educational experience, which undermines its validity. Having established foundational knowledge of constructivism and its potential shortcomings, let us consider *Microsoft Flight Simulator* through a constructivist lens.

Bogost argues that “a constructivist might understand” this game “as a game that teaches professional knowledge through ‘performance before competence,’” which is a “concept of pedagogical apprenticeship” (Bogost 239). In other words, a constructivist would appreciate *Microsoft Flight Simulator* because its effective simulation allows the user to learn *by* doing, rather than learn before doing. When analyzing video games through a constructivist lens, we uncover “the abstract systems that underlie them,” whereas behaviorists tend to focus strictly on what knowledge is imparted through the procedural representation of the game (Bogost 239). To this end, let us consider a very different EVG, *Lemonade Stand*.

In *Lemonade Stand*, the player operates a lemonade stand for a set amount of days, with the objective of optimizing your profits (coolmath.com 2021) Each day, the player views the weather report and purchases the amount of lemons, sugar, ice, and cups that they see fit. They can also change the recipe of the lemonade and price of a cup every day. During the day, the player watches customers pass by the lemonade stand and express feedback on their product, saying things like “yuck,” “more ice,” and “mmm” (coolmath.com 2021) At

the end of every day, the player receives feedback regarding their performance that day. A behaviorist would determine that this game teaches its player the value of the dollar, in that the player must coordinate the recipe to the purchases of ingredients, in anticipation of how much lemonade they expect to sell. A constructivist, on the other hand, would determine that this game is a successful EVG in that it teaches its player basic economics, in that supply and demand vary based on external factors, like the weather and ingredients spoiling. When playing this game, I learned that on hotter days I should purchase more ice and increase the amount of ice in my recipe. I also increased the price in anticipation of the likelihood that people are more likely to want lemonade on a hot summer day. Thus, a behaviorist would view this game as a more mathematical EVG, imparting knowledge on the player regarding their recipe and supply. A constructivist, on the other hand, would view this game as an economics EVG, introducing the player to the fluctuating market and teaching them how to respond accordingly. In summation, a behaviorist "ascribes a singular, rationalist approach upon the content of videogames" (Bogost 240). Conversely, constructivists tend to "divest" the "specificity of a particular video game in favor of the general, abstract principles it embodies" (Bogost 241). This is to say that when determining the efficacy of an EVG, behaviorists focus on the procedural representation of a game, whereas constructivists center the procedural abstraction. While these interpretations seem to be at odds, Bogost offers a happy medium between the two, claiming that a game's procedural representation and its underlying rhetoric are equally important.

Bogost shows the impartial importance of both procedural representation and rhetoric through game designer Raph Koster's "hypothetical reskinning of the classic puzzle game *Tetris*" (Bogost 242). In Koster's reimagination of the procedural representation of *Tetris*, he has "the player dropping innocent victims down into the gas chamber, and they come in all shapes and size" much like the varying *Tetris* blocks that spawn and sink to the bottom of the

board (Bogost 242). He continues, “as they fall to the bottom, they grab onto each other and try to form human pyramids to get to the top of the well. Should they manage to get out, the game is over and you lose. But if you pack them in tightly enough, the ones on the bottom succumb to the gas and die” (Bogost 242-243). In Koster’s reconception of *Tetris*, he maintains the same procedural rhetoric and perverts the procedural representation to simulate a horrific gas chamber. Through this example, we recognize the inseparability of a game’s procedural rhetoric and representation. This imagined reskinning of *Tetris* serves to show how a game’s procedural representation serves the game’s procedural rhetoric. This is to say that a player’s experience and interpretation of *Tetris* would be vastly different if its procedural representation were something so horrific. The procedural representation of a game compounds a game’s underlying rhetoric. As Bogost puts it, “the coupling of abstract processes to particular topics produce particular meanings that represent particular positions” (Bogost 243). Therefore, when designing an EVG, one must consider how the abstract process and particular topics compound a particular meaning for the player. Conclusively, a game’s procedural representation and rhetoric are paramount in the construction of an effective EVG.

Playability Guidelines for Educational Video Games serve to highlight this dynamic in a much more simplistic manner. This literature review emphasizes that the “two fundamental pillars” of an EVG are “fun and education” (Ibrahim). *Playability Guidelines* asserts that many EVGs “‘have failed either because games designed to educate do not engage their intended audience, or because truly engaging games do not provide enough educational value’” (Ibrahim). This is a common criticism of edutainment, in that it is often neither educational nor entertaining due to their inability to balance these aspects. Much like the behaviorist-constructivist dichotomy, this literature review reiterates the importance of balancing the educational content (procedural rhetoric) with engaging design (procedural

representation). In short, an EVG must consider its educational contents and principles of engaging design with equal priority. To demote either aspect will certainly impact the success of an EVG. Having established how *Playability Guidelines* recognizes the significance of a balanced design dynamic, let us discuss the importance of player experience.

Surely, player experience is no easy quality to measure. Bogost acknowledges the inherent "subjectivity of representation" that varies under "different player contexts" (Bogost 239). This is to say that while a game may intend to represent something in a particular way such that it imparts some particular educational rhetoric unto the player, the interpretation of the game's representation is placed in the hands of the player. *Playability Guidelines* acknowledges the complexity of quantifying player experience, citing how "'difficult'" it is to "'obtain knowledge about what players did when playing the game, and how meeting different game design elements affected their experience of interacting with the game'" (Ibrahim). Indeed, quantifying the player experience of a game is a difficult endeavor, however, playability is no less paramount.

In light of this importance, *Playability Guidelines* defines a set of principles by which to measure playability. The set of principles outlined in *Playability Guidelines* is nothing short of comprehensive, and at times, redundant, so let us consider some of the most essential principles. First, it suggests "satisfaction" as a marker for playability, defining it as "the gratification […] derived from playing" the game (Ibrahim). While this quality is intangible, it is something that should be prominently considered in the design of a game. Indeed, this quality reflects that a game should be designed such that the player feels rewarded for their efforts. Next, it suggests learnability as an important aspect of the player's experience. Learnability is "the player's capacity to understand and master the game system and mechanics," which is to say that learnability measures the ease in which the player learns to play the game (Ibrahim). Following learnability, *Playability Guidelines* defines effectiveness

as a pillar of the player's experience. Effectiveness is defined as the conjunction between "fun and learning," offering the players "a new experience [...] while they achieve the game's various objectives and reach the final goal" (Ibrahim). Effectiveness reiterates the importance of making an EVG an engaging game to play for anyone, not just those who seek to learn the game's educational content. Alongside effectiveness lies immersion, which is a quality that is paramount in any video game's design, not just for EVGs. This review defines immersion as "the capacity of the EVG contents to be believable, such that the player becomes directly involved in the virtual game world" (Ibrahim). Consider *Curse Reverse*, a math EVG whose procedural representation is to traverse tombs and collect items to reverse a curse. Its educational objective, or procedural rhetoric, lies in its requirement of the player to scale variables such that the player can pass through the tomb and collect the item. This game has effective educational design principles, in that it constructs an environment that exhibits the utility and function of scalar variables. However, the game world is not intrinsically immersive. Therefore, when designing a captivating EVG, one must concern themselves with designing a holistically engaging game. Next, we must consider motivation, which encompasses the "set of game characteristics that prompt a player to realize specific actions and continue undertaking them until they are completed" (Ibrahim). This is to say that the game must be continuously engaging, such that the player is motivated to continue tackling the feats the game presents. In consideration of an EVG, we recognize that "the motivation to play produces an indirect motivation to learn" (Ibrahim). Lastly, we have educatability, which is defined as "the video game's capacity to include […] educational content and to provide it to the player in an active manner" (Ibrahim). This principle of player experience seems to address a common fallacy of EVGs that construct a fun player experience but have insignificant educational value. In summation, *Playability Guidelines* qualifies a set of

principles that contribute to a game's player experience: satisfaction, learnability, effectiveness, immersion, motivation, and educatability.

Through this comprehensive review, we have established a sufficient wealth of knowledge regarding EVGs and effective design. By considering Bogost's presentation of behaviorist and constructivist educational ideologies, we have a framework through which to consider a game's educational design. Moreover, we have recognized the importance of balancing a game's procedural rhetoric with its procedural representation. Lastly, we have determined what principles of game design to consider in pursuit of designing a game in light of the player's experience. As we navigate the design of my EVG, let us consider these characteristics throughout such that we can determine its efficacy and potential for improvement.

Chapter 2. Methods

To create this game, I used the free and open source Javascript library p5.js, a creative coding library with built-in tools to both modify the DOM and enable drawing functionality within a web browser. Before we can dive into how p5.js extends JavaScript's functionality, let us first establish how JavaScript works in conjunction with CSS aand HTML to construct a webpage.

An HTML file is written in Hypertext Markup Language (HTML), which is a declarative programming language that formats the structure and contains the content of a webpage (W3Schools 2021). contains two major sections: the head and the body. The head contains the meta information of the HTML page, as well as the necessary assets needed to construct the page (i.e. font or images). Furthermore, the HTML head often includes links to the CSS and JavaScript files. Next, the body section of an HTML file contains the actual content of the page. The body contains the content and order in which the content is displayed. A single piece of content in the body of an HTML page is called an element. Thus, with a foundational understanding of an HTML file, let us establish the role CSS plays in the construction of a webpage.

A Cascading Style Sheet (CSS) is responsible for the style of the webpage. Similar to HTML, CSS is also declarative. To define stylistic characteristics of an element on your HTML page, you specify what element(s) you want to modify, and define any stylistic choices for those elements, such as font size, color, or positioning (W3Schools 2021). In short, the CSS file for a webpage applies styling to the HTML content.

Lastly, JavaScript is a multi-paradigm language that implements dynamic functionality to a webpage (W3Schools 2021). JavaScript is a powerful tool for web programmers because it can access and augment the contents of the HTML and CSS files. In other words, you can create a complex website using just JavaScript with very little in your

HTML and CSS files. Having established the basic structure of web programming, we can now examine the structure of my p5.js project.

My p5.js project includes an assets folder, an index.html file, a style.css file, and a sketch.js file. My sketch.js file contains the vast majority of the relevant code for my project. My index.html file is rather sparse, and the only content in the body of my HTML file is a script element containing my sketch.js file. My style.css file contains some style settings for the canvas, specifically for button and input elements. As I said before, the sketch.js file contains all the relevant code that controls the game itself, which is available to access via this link https://github.com/allisonknox1393/Transformers.

Let us first review how the game works: the user is presented with a conic section represented on a graph. They are expected to construct the equation for that conic section based on its graphical representation. First and foremost, I started constructing the function that would be responsible for drawing the x-y coordinate plane to the screen. I created a function called drawGrid that took in two arguments: the midpoint of the width of the viewport and the midpoint of the height of the viewport. This function then executes two for-loops. The first for-loop iterates ten times and draws ten horizontal rectangles and ten vertical rectangles to the screen using the built-in rect function. The rect function expects four arguments: the first two give the (x, y) coordinate of the center of the rectangle, and the last two arguments provide the width and

```
noFill();//rectangles drawn without fill so they overlap
let side = w / 2; //decrementation variable = 500
for (let i = 1; i <= 10; i++) {
  //if we are drawing a rectangle at the origin, increase
  //the strokeweight so that the x and y axes are distinctly visible
  if (side - (i * side) / 10 == 0 || side - (i * side) / 10 == side)
  {
    strokeWeight(3);
  } else {
    //otherwise maintain the strokeweight
    strokeWeight(1);
  }
  //draws the rectangles that decrease in height (y-axis)
  rect(centerW, centerH, side, side - (i * side) / 10);
  //draws the rectanlges that decrease in width (x-axis)
  rect(centerW, centerH, side - (i * side) / 10, side);
}
rect(centerW, centerH, side, side);
```

Figure 1. Code snippet from drawGrid

height of the rectangle. To display twenty rectangles such that they appear as a single grid with the dimension 1000x1000 pixels, I draw ten rectangles that decrease in their height by 50 pixels and ten rectangles that decrease in their width by 50 pixels. By using the noFill command, it appears as a 20x20 matrix of uniform squares.

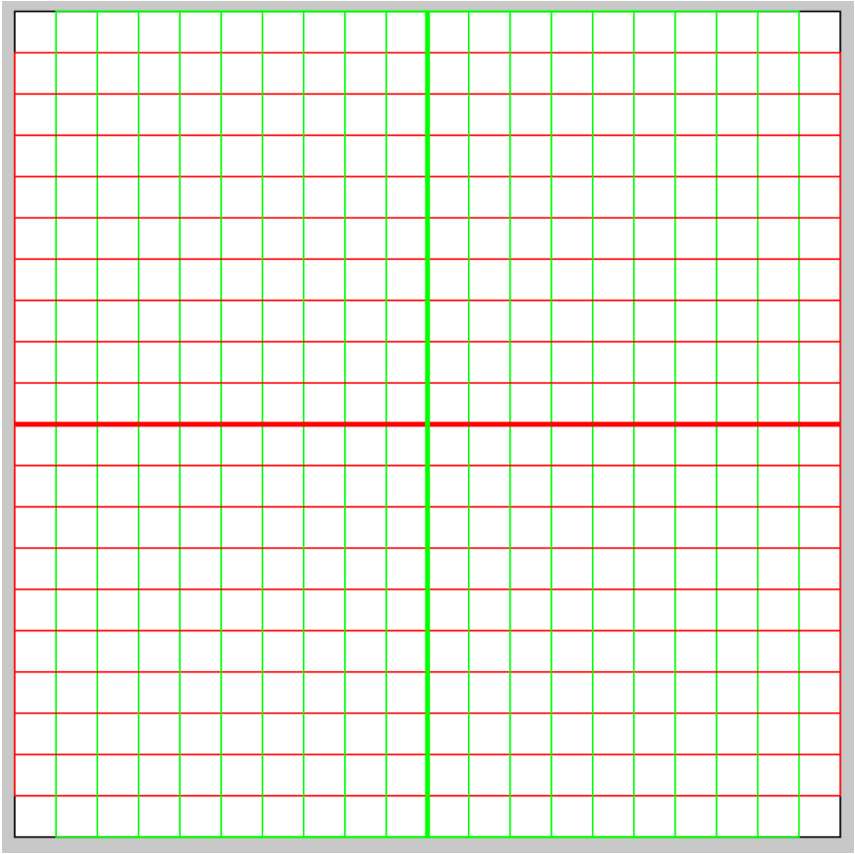

Figure 2. Display of grid

Thus, we are drawing twenty total rectangles to the screen that intersect such that it appears as if there is a 20x20 matrix of squares such that the side of each square is 25 pixels. Once we exit the for-loop, we draw one last rectangle with width=500 pixels and height=500 pixels such that it is the last square needed for the grid. Below is a depiction of this grid in which the red rectangles represent the rect function calls whose height decreased throughout the for-loop, and the green rectangles represent the rect function calls whose width decreased throughout the for-loop. The black rectangle is the rect function call that is executed once the for-loop has finished executing.

After the grid portion is taken care of, the function continues and enters a new for-loop that is responsible for labeling the axes. For the axis labels, I imported a separate font and used an array of strings that contained all integers from -10 to 10. Then, on each iteration through the for-loop, I display the next string element in the array at a certain position. This was a bit trickier because of how the text function works in p5.js, primarily because longer string lengths (i.e. “-10” versus “10” or “-9” versus “9”) display differently based on their length. Through some careful maneuvering, I was able to effectively display all of the axis labels. Therefore, with the drawGrid function, we have the most foundational element to the game. While creating this function took a lot of trial and error, I am extremely pleased with its aesthetics. Next, let us discuss how I navigated the construction of the levels and the mechanics of switching levels.

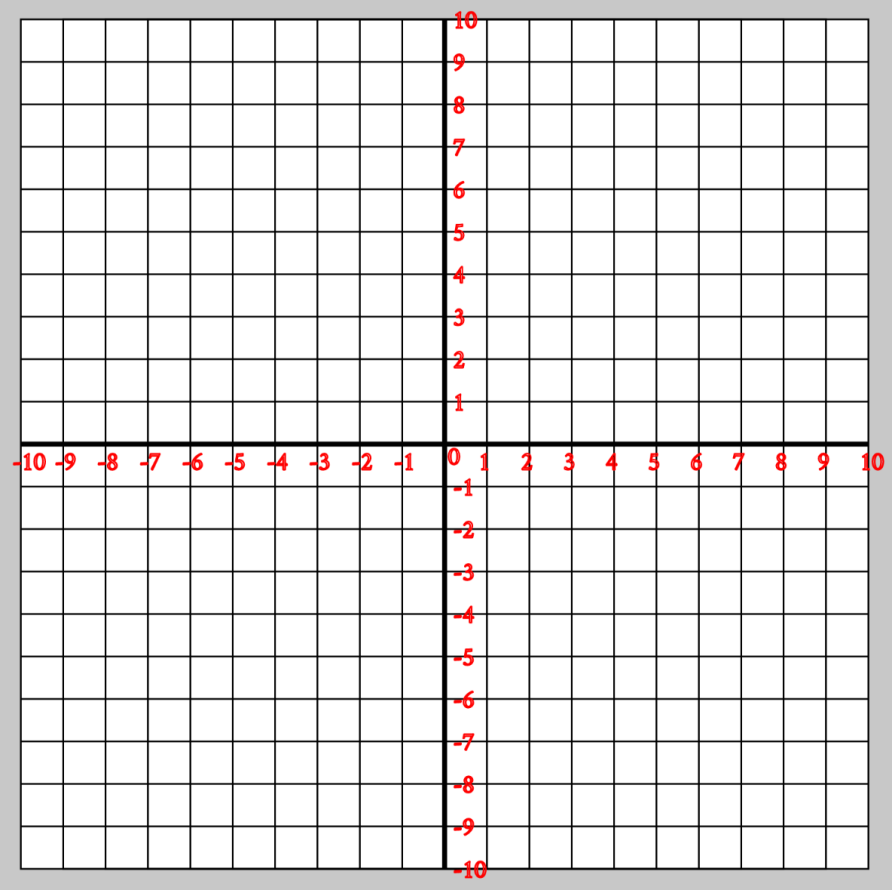

Figure 3. Displayed grid with axes labeled

In short, my goal for the mechanics of switching levels was to create one global boolean variable per level, and then those booleans could serve as light switches for the levels. For example, if level one's global boolean is currently false, then that level is not displayed. Once some trigger or condition or event happens, it flips the switch, sets the current level's boolean to false and the next level's boolean to true. Then the next level would be displayed. To implement this in code, I declared a series of global variables at the top of the file. I initialized all of the global booleans in the setup function, which is a function that runs once at the start of the program. In setup, every global boolean was set to false, except for the start screen's boolean, because that is the screen displayed when the game starts. Next, I utilized the draw function, which is a built-in function in p5.js that runs after setup, and loops every frame of the program. In draw, I have an if-else if block that is constantly looping to check which level should be displayed based on that level's corresponding boolean variable's value. If a current level's boolean is set to true, then we call that level's corresponding function which displays all the necessary elements. Then, within that function, there is some trigger (i.e. a mousePressed event on a button) that then calls a separate function that flips off the current level's switch and flips on the next one's. In short, using simple coding structures like global boolean variables and mousePressed events, I can control the flow of the game with ease.

```
global boolean startScreen;
global boolean levelOne;
global boolean levelTwo;

function setup(){
startScreen=true;
levelOne=false;
levelTwo=false;
}

function draw(){
      if(startScreen){
            displayStartScreen();
      }
      else if(levelOne){
            displayLevelOne();
      }
      else if(levelTwo){
            displayLevelTwo();
      }
}
function display startScreen(){
      if(start level one button is pressed){
            startScreen=false;
            levelOne=true;
      }
}
```

Figure 4. Pseudocode for methods described

With the basic mechanics of the game down, such as the grid and the levels, I was then able to start implementing the more complex functionality of the game: parsing the player's inputted equation and displaying it as a conic section on the grid. To do this, I first had to take in some user input. I thought this would be a rather simple endeavor: just create an input element and a corresponding button. Then the user could type an equation in the input element and press the button to submit it. However, this did not work. When I was testing my program as a user, I was not able to see the text that I was typing in the input element. Since the input element is being created in a level's function, and that function is being called every frame of the program, then a new input element is being created every frame of the program. I thought that I had hit a wall because I needed the draw function to determine which level should currently be displayed, but draw's permanent loop was causing a core issue in my program. I overcame this issue when I discovered the noLoop and redraw functions that p5.js offers. According to the p5.js reference, if you call the noLoop function at the end of setup then it augments draw such that it does not loop every frame of the program. Then, I could use the redraw function which is a function that updates draw exactly once when redraw is called. These two functions were exactly what I needed because I only needed draw to update when an event or trigger occurred. So I put noLoop at the end of setup and called redraw on every mousePressed event handler function such that draw would only update when necessary. With this implemented throughout my code, I was able to successfully utilize the input and button elements within a function without any issues. Now that I was able to access the user's input, I could start trying to parse their inputted equation.

To parse the user's equation, I had to learn how to use Regular Expressions (RegEx). RegEx is a string pattern that is used to match character combinations in strings. I used RegEx to match character combinations such that I could extract all of the relevant information from a string containing an equation of a circle. To do this, I used the match

function which takes in a regular expression and a string and returns an array of strings in which the elements in the array are the matching groups. First, we will address parsing the equation of a circle. Consider the vertex form of the equation of a circle, given by $(x-h)^2+(y-k)^2=r^2$ such that (h, k) represents the center of the circle, and r represents the radius. Given the center of a circle and its radius, I can use that information to draw the circle to the screen. Therefore, my match function should return an array that contains h, k, and r^2.

However, this means that for the RegEx to match the string input such that I could extract all the relevant information, I had to require that the user input the equation in vertex form every time. Regardless, this made sense from both an educational standpoint and made the parsing function much simpler. Consider the following code snippet from the function that parses the equation of a circle from an input element.

```
160   let equation=eq1.value(); //inputted equation
161   let circle='\\(x(\\+\\d|\\-\\d)\\)\\^2\\+\\(y(\\+\\d|\\-
    \\d)\\)\\^2\\=(\\-\\d|\\d)'; //regex
162   let returned= match(equation, circle); //stores the array
163   let h=parseInt(returned[1]); //int value of h
164   let k=parseInt(returned[2]); //int value of k
165   let r=parseInt(returned[3]); //int value of r
166
167   //call drawCircle and pass h, k, and r
168   drawCircle(h, k, r, centerW, centerH, true);
```

Figure 5. Code snippet from parseCircleInput

As you can see, in line 162 we store the String array that the match function returns in a variable. Then, we know that the element in the returned array with index 1 is always going to correspond to the value h, and that the element with index 2 is going to correspond to k, and that the element with index 3 is going to correspond to the radius. Then, using that information I can call the drawCircle function and effectively draw the user's inputted equation to the screen.

The drawCircle function is responsible for displaying the accurate graphical representation of a circle given the provided information about its equation. The drawCircle function takes in the following parameters: three integers h, k, and r, and two integers that

correspond to the midpoints of the current width and height dimensions of the viewport. Then, inside the function, I call the translate function which translates the origin (0,0) of the screen from the top left corner to the x and y coordinates passed in as arguments. Using the translate function, I translate the origin to the center point of the grid, such that it corresponds to the grid's origin. Then, I can use the circle function to draw a circle to the screen. The circle function takes in three arguments: the x and y-coordinates of the center of the circle, and the diameter of the circle. Since I am given the (h, k) center point from the equation, I can draw the circle to the screen provided that I interpret the h, k, and r correctly. If the equation of a circle has a positive h value, that means that the x-coordinate of the circle's center is at $-h$. Conversely, if the equation of a circle has a $-h$ value, that means that the x-coordinate of the circle's center is at h. Then, from a programming standpoint, this means that given an argument, h, I need to pass in $-\mathrm{h} * 25$ as the x-coordinate of the circle's center in the actual circle function call. The value 25 comes from the fact that the width of each unit on the grid is 25 pixels. Next, let us consider the y-coordinate of the center of a circle, given by k. If k is positive in the equation of a circle, then that means that the y-coordinate of the center point of the circle is at k, and if it is negative then the y-coordinate is at $-k$. From a programming standpoint, we need to pass in $-\mathrm{k} * 25$ as the y-coordinate of the circle's center in the circle function call. This is because, with the origin translated to the center of the grid, for positive k to display on the positive y-axis, we need to pass in a negative value such that the circle is positioned further up on the grid. Conversely, for $-k$ to display on the negative y-axis, we need to pass in a positive value such that the circle is positioned further down on the grid. Lastly, let us consider how we interpret the argument r. Since r represents the value r^2 as it is represented in the equation, we need to pass in $\sqrt{\mathrm{r}} * 25$ for the diameter argument in the actual function call. Thus, we have accurately interpreted the relevant information from

the parseCircleInput function such that we can display any equation of a circle graphically, as represented in the code snippet below.

```
function drawCircle(h, k, r, originX, originY, user) {
  let unit = 25; //unit variable to scale the circle accurately
  push(); //limits translation matrix to this function
  translate(originX, originY); //translate to origin of grid
  noFill();
  //centerX=-h*25, centerY=-k*25, diameter=sqrt(r)*25
  circle(unit * -h, unit * -k, sqrt(r) * unit);
  pop();//end translation
}
```

Figure 6. drawCircle function

Next, let us consider how we parse the inputted equation of a parabola. In vertex form, the equation of a vertical parabola is given by $y = a(x - h)^2 + k$, in which a represents the slope of the parabola and (h, k) represents the coordinates of the parabola's vertex. The algorithm for the parseParabolaInput function is similar to the parseCircleInput function we discussed previously. There are two notable differences in this function's algorithm, the first of which is that for the matched element a in the returned array, I had to consider the case when a is a fraction. Of course, this could have been true for the circle function, but I designed all the circles in the levels such that their center points were always integers. With parabolas, I often needed to scale their slopes to fractions so that they could serve the picture constructed by the conic equations on the screen (for example, I display parabolas that look like eyebrows in level 6). To implement the fraction case in code, I used an if statement with the boolean method includes to determine if there was a "/" character in the returned element corresponding to a. If this if-statement was satisfied, I used the split function. The split function takes in a string and a delimiter (a valid RegEx string) and returns an array of strings such that the elements of the array are substrings found between each delimiter. In short, the delimiter passed through the function serves as a character that marks a boundary between each piece of the string. Therefore, if I used the split function I could

determine the numerator and denominator of the fraction a. Consider the following snippet of code from the parseParabolaInput function that showcases the implementation of the methods described.

```
  let parabola =
    "y\\=(\\d|\\-\\d|\\d\\/\\d|\\-\\d\\/\\d)\\(x(\\+\\d|\\-
\\d)\\)\\^2(\\+\\d|\\-\\d)";

  let returned = match(equation, parabola);
  let a = returned[1];

  //if a is a fraction
  if (returned[1].includes("/")) {
    //split a based on / to access numerator and denominator
    let newFraction = split(returned[1], "/");
    let num = newFraction[0];//numerator
    let denom = newFraction[1];//denominator
    a = parseFloat(num / denom); //parsefloat of fraction
  }
```

Figure 7. Code snippet from parseParabolaInput

The other notable difference in this function's algorithm is that for a parabola equation, the user is expected to input bounds for the domain of the parabola. This, again, is because the conic sections make up an image on the screen such that their domain needs to be restricted. This was a simple implementation in which I created two separate inputs for a parabola equation, corresponding to the lower and upper bounds. With the implementation of the parseParabolaInput function, I could then create the drawParabola function.

The drawParabola function actually proved to be rather difficult to implement. Algorithmically, I knew I needed to interpret the arguments from the user's equation, specifically $a,\ h,\ k$ and the domain, such that it displays the accurate representation of the parabola to the screen. However, there is no built-in parabola function like there is for a circle. For unique shapes in P5.js you can utilize the beginShape and endShape functions, which sandwich a series of valid vertex commands to construct a shape. The vertex command expects two arguments, corresponding to the x and y coordinates of the vertex. When used to construct a shape, beginShape stores the coordinates of all the vertex commands until

endShape is called. Then, it connects each vertex with a straight line. Therefore, since a parabola is symmetrical about its vertex, I realized that I could create the parabola with two shapes that represent the left and right halves. Each shape would be constructed with thousands of vertices such that it would appear as a curve. To do this, I created a for-loop for each shape that increments i by 0.01 on each iteration. The first for-loop created the vertices for the left half of the shape, that is, the portion of the parabola containing points from h to the lower bound of the domain. The second for-loop created the vertices for the right half of the shape, or the portion of the parabola containing points from h to the upper bound of the domain. Before each for-loop, I used the translate function to translate the screen's origin points to the vertex of the parabola. Then, if the scalar value a was negative, I used the rotate function so that the parabola would face down. Then we can iterate through the for-loops and create the shapes. Within each iteration, we initialize an x and y variable that correspond to the parabola equation, $(x, f(x))$, and pass in x and y as the arguments for the vertex function call.

```
  beginShape();
  let lowerBound=0;
  //determines the exit condition of the for-loop that creates the
shape using h and lower arguments
  for(let i=-h; i>lower; i--){
    lowerBound+=1;
  }

  //creates the shape from [h, lower]
  for(let i=0; i<=lowerBound; i+=0.01){
    let x=i*unit; //x
    let y=(i*i)*unit*a; //f(x)
    vertex(x, y);
  }
  endShape();
```

Figure 8. code snippet from drawParabola function

Perhaps the hardest part of constructing these for-loops was ensuring that the domain worked in all cases. This was difficult and perhaps could have been done without a for-loop, but the separate lowerBound for-loop was the solution that I landed on. While it is perhaps not the

most optimal runtime, the drawParabola function is effective and accurate in displaying a parabola to the screen given values from its equation.

After I had implemented the functions that parse equations, as well as the functions that then display the corresponding conic section, I had completed the bulk of the difficult-to-implement portions of the game. From here, I just focused on designing the levels and adding extra functionality to the game to improve the player's experience. For example, I implemented a button on every level that toggled a display setting for the conic sections. If they pressed this button, it would draw the center point and radius for the circles on the screen, as well as the vertex and potential x or y-intercepts for the parabolas on the screen. While it does not label these points, this setting is intended to improve the visualization of each conic section so that it would make creating the corresponding equations easier.

Chapter 3. Results and Analysis of the Game

Previously, we discussed two educational philosophies: behaviorist and constructivist. Recall that behaviorists believe that education happens through repetition and reinforcement, while constructivists believe that education is constructed individually-- through a learner's experience with their environment. In light of these ideologies, I would consider this game to follow a behaviorist approach as an EVG. This is because the game imparts knowledge to the user through repetition and reinforcement. Through increased complexity and quantity of conic sections displayed at a given level, the user builds on their previous knowledge and experience with the game. Moreover, this EVG falls more into the behaviorist category because a behaviorist focuses on the procedural representation of the game. The procedural representation of this game does successfully construct a microcosm environment that is modeled like a classroom, which, Bogost argues, is how a behaviorist would design an EVG. In this sense, the game is effective in imparting knowledge to the user. However, this game's procedural representation and implementation of behaviorist ideology certainly seem to affect its playability.

Let us critique this game in terms of the principles of design laid out in Playability Guidelines. As Playability Guidelines makes clear, the hardest part of designing an engaging EVG is striking the balance between a game that is intrinsically fun and educational, without falling too far into either category. Indeed, most EVGs are less than effective either because they are very fun but not educational enough, or very educational and not fun enough. I would argue that this game falls on the side of being very educational, but not engaging enough. Indeed, the design of the game relies heavily on mathematical principles like x-y coordinate planes. Quite frankly, the visual design of the game looks a lot like a homework assignment from an Algebra II class. Indeed, this game lacks a lot of the principles that Playability Guidelines defines. For example, this game is not very immersive. Recall that

immersion is defined as “the capacity of the EVG contents to be believable, such that the player becomes directly involved in the virtual game world” (Ibrahim 25). Common tools of immersion in videogames include music or animations. This game lacks characteristics that make the gameplay enjoyable and immersive for the user, such that they are hunched over in anticipation of what comes next. Moreover, this game is not designed in a way such that it motivates the player. Motivation is an important aspect of playability; it describes the characteristics of a game’s design that “prompt a player to realize specific actions and continue undertaking them until” the game’s objectives “are completed” (Ibrahim 25). Indeed, an EVG can be very effective if “the motivation to play produces an indirect motivation to learn” (Ibrahim 25). I would argue that the motivation to play this game is not produced by the game itself, but rather someone would be motivated to play this game because they are strictly motivated to learn. Certainly, the design of this game could be improved such that the player’s experience is more rewarding.

While I am still proud of the game I made, I will be the first to admit that it could use some improvements. First and foremost, I would attempt to improve the immersive characteristics of this game. To do so, I would implement a soundtrack for the game, as well as other sound effects so that the gameplay feels more responsive and interactive. Moreover, I could introduce an animated character, like a mascot, who follows the player throughout the game and reacts to the gameplay. Of course, because of the repetition-reinforcement ideology behind the game design, this game’s immersive potential is limited. Regardless, there are still improvements that could be made to make the game a more immersive and rewarding experience for the player.

Moreover, to improve the game’s motivational aspects, I would implement the creator mode. The creator mode is a game mode that unlocks once the player has completed all of the levels, in which the player can drag and drop conic sections onto the grid to create their art.

Then, they would enter back into the level mode such that they could determine the equations of each conic section and complete the image they outlined. In conjunction with the creator mode, I could implement a rewards system in which a user earns some type of currency if they determine an equation in fewer tries. This would motivate the player to play the game thoughtfully. I could implement some kind of shop in creator mode, in which the player can exchange their currency for colors and patterns that they can use in their own art. I feel that this would have benefitted the game significantly, as it would provide an end goal for the player to traverse the levels. Furthermore, it would change the overall goal of the game from learning conic sections and linear transformations to creating your art with conic sections. This would significantly affect the player's experience, as it would reframe the motivation from learning for learning's sake to learning so that you can create something yourself. With this implementation, the game's rhetoric would strike a better balance between behaviorism and constructivism.

Chapter 4. Conclusion

I set out to make an effective EVG that helped the player learn about conic sections and linear transformations. By using P5.js, I was able to develop an HTML game that displayed conic sections to a grid, and prompted the user to determine the corresponding equation. This game is certainly educational, as its design principles are grounded in behaviorist ideology that asserts an effective EVG creates a microcosm of an educational environment and reinforces knowledge through repetition. However, the efficacy of this EVG overall suffers due to its negligence of the player's experience. This game could improve its playability with some implementations—like the creator mode—that make the experience more immersive and motivating. Moving forward, I plan to implement some of the design characteristics discussed in the results to improve the playability and overall efficacy of this EVG.

Bibliography

Bogost, Ian. “Procedural Literacy.” *Persuasive Games: the Expressive Power of Videogames*, by Ian Bogost, The MIT Press, 2010, pp. 233–260.

CSS Introduction, Resfnes Data, 2021, www.w3schools.com/css/css_intro.asp.

“Curse Reverse Game.” MathSnacks.org, NMSU Board of Regents, 2009, mathsnacks.com/curse-reverse.html.

“HTML Introduction.” Introduction to HTML, Refsnes Data, 2021, www.w3schools.com/html/html_intro.asp.

Ibrahim, Amer. Review of *Playability Guidelines for Educational Videogames: A Comprehensive and Integrated Literature Review, International Journal of Game-Based Learning*, 2012, pp. 19–38.

JavaScript Introduction, Resfnes Data, 2021, www.w3schools.com/js/js_intro.asp.

JavaScript HTML DOM, Refsnes Data, 2021, www.w3schools.com/js/js_htmldom.asp.

“Lemonade Stand.” *Cool Math Games*, Coolmath.com LLC, 2021, www.coolmathgames.com/0-lemonade-stand.

MartÍ-Parreño, José, et al. “Students' Attitude Towards the Use of Educational Video Games to Develop Competencies.” *Computers in Human Behavior*, vol. 81, 2018, pp. 366–377., doi:https://doi.org/10.1016/j.chb.2017.12.017.

“p5.Js Reference.” *Reference | p5.Js*, 2019, p5js.org/reference/.

Su, Francis. *Mathematics for Human Flourishing*. Yale University Press, 2021.