Unit 1
Videogames and Coordinate Planes

Unit Overview

Students discuss the components of their favorite videogames, and discover that they can be reduced to a series of coordinates. They then explore coordinates in Cartesian space, and identify the coordinates for the characters in a game at various points in time. Once they are comfortable with coordinates, they brainstorm their own games and create sample coordinate lists for different points in time in their own game.

Agenda

5 min	Introduction and Expectations
10 min	Dissecting a Demo
15 min	Coordinate Planes
10 min	More dissection
15 min	Brainstorming!
15 min	Intro to Racket and Numbers
20 min	Circles of Evaluation
5 min	Review

Learning Objectives:

- Learn to work as a team
- Learn the Cartesian coordinate system
- Make the jump from PEMDAS to function composition, when thinking about arithmetic
- Students identify the order of operations for complex arithmetic expressions through diagramming (circles) and solving (the numerical answer)

Product Outcomes:

- Reverse-engineer a simple, side-scrolling game
- Brainstorm a videogame
- Identify coordinates of various images in a screenshot
- Converting between multiple representations of arithmetic (Conventional, Circles of Evaluation, and Racket)

State Standards See our Standards Document provided as part of the Bootstrap curriculum.

Length: 90 minutes

Materials and Equipment:

- *Student workbook folders - in pairs! - with names on covers.*
- *Computer, connected to a projector, with "NinjaCat" preloaded [NinjaCat.rk from source-files.zip | WeScheme]*
- *Pens/pencils for the students, fresh whiteboard markers for teachers*
- *Cutouts of NinjaCat, Dog and the Ruby*
- *Class posters (List of rules, language table, course calendar)*
- *Language Table (see below)*

Preparation:

- *Distribute workbooks, pens, arrange chairs so students are in pairs*
- *Set up student machines and projector, and check to make sure it all works!*
- *Post the agenda, rules, calendar, basic skills, language table, etc*

Introduction and Expectations (Time 5 minutes)

- Welcome to Bootstrap!
- *Introduce the teaching staff. Give some background: age, where you're from, something surprising about yourself, favorite food, etc. Anything to allow kids to connect. Ask kids for their names!*
- *I want to set some expectations here, to make sure we're all on the same page.*
- In Bootstrap, you'll be programming in a language called *Racket*, which is used at universities and colleges around the world. You'll be doing the same kinds of programming that they do, using the same techniques and similar tools.
- Because this is the same material that they teach in college, you'll need to approach the material the way college students do. What do you think that means?
-

  Exercise:

  Take 2-3 suggestions. Some samples you'll want to talk about: making mistakes it okay! Getting the answer quickly isn't as important as taking the time to think it through! Working in teams is a good thing!

- *Okay, so here are the rules: (point to a visual while going over this)*
  - *Unless we say otherwise, raise your hand to speak*
  - *The computers are for programming, not for Facebook, chatting, email or playing around*
  - *If you're not programming, the computer monitors need to be OFF. When we say "monitors off," everyone should shut off the monitors, stop talking, and give their attention to the teachers. Let's try this one out - everyone make sure the monitors are on ... now start talking ... louder! ... MONITORS OFF! Repeat as necessary.*
  - *No one in this class is allowed to put each other down. I don't ever want to hear anyone call somebody else "stupid," "idiot", or anything else. Don't tell each other to shut up, and be respectful to everyone. Is that clear?*
- When it comes time to program your games, you'll be working in pairs - you and a friend. But during class, there are a lot of activities and games we'll be playing where you and your partner will be part of a group.
- Right now, all of you are sitting with your group, clustered together around the classroom. When it comes time for team competitions or group activities, you'll be working with the people sitting nearby.
- Depending on how well you work together, we'll be able to make some adjustments from class to class.

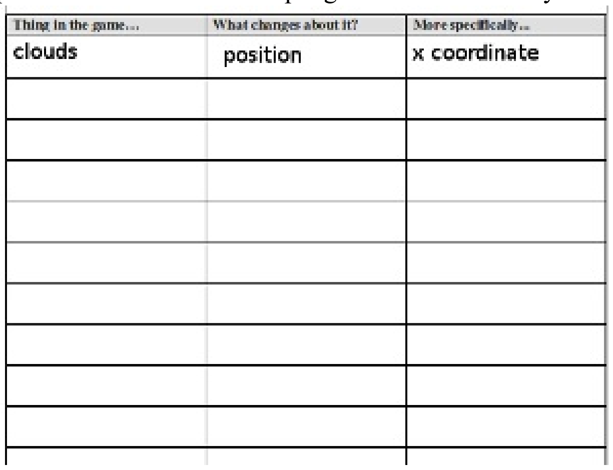
- Exercise:

  Assign group names (tip: use names that have something to do with computer science, or college, etc. Do NOT use ordinal names like "Team A" and "Team B" or "Team 1" and "Team 2" – who wants to be stuck on Team B?)

- We'll be using Racket to program our videogames. As with any language, you'll need to learn the basics before we get into the complicated stuff! While you'll be thinking about and designing your videogames for the entire class, we're going to start out with the basics and build our way up from there.
- For example, videogame programmers need to be really good at geometry in order to draw things at precise locations onscreen, and good at algebra to write functions that cause things to move, transform or explode in the game.
- In each lesson, we'll talk about a new part of the videogame, the math concepts behind it, and then how to program those concepts into your game.

Dissecting a Demo (Time 10 minutes)

- Soon you're going to be designing your own games, but it's important to know how a simple game is built before you start designing your own.
- Let's take a look at a real game, which I've made using the same language and tools that you'll be learning. When you're done with this class, you'll be able to make something similar.
- *Demo: Show the kids Ninjacat. Be sure to end on a frozen frame of the game, so the whole class can see the same image.*
- This game, like many others, is made up of *characters*, each of which has it's own behavior. The Ruby moves from the right to the left, as does the Dog. NinjaCat only moves when you hit the arrow keys, and she can move left, right, up and down. We can figure out how the game works by first understanding about how each character works.



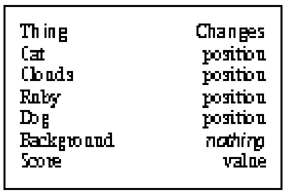
Thing in the game...	What changes about it?	More specifically...
clouds	position	x coordinate

- Turn to Page 1 in your workbook. Here we have a table, which we'll use to reverse-engineer NinjaCat and see how it works.
  Raise your hand if you can tell me one thing that you saw in the game. Wait until a few hands are up, then call one. Excellent! So we can write that down in our first column.

- Exercise:

  In your groups:
  Take one minute to fill out thge first column with a complete list of all things in the game.
  Your group will get a point for each thing they can find. Everyone in your group should have this list written down - not just one person! If even one person in your group hasn't written it down, the group doesn't get the point! GO!

- *During the minute, walk around and see how groups are doing. This is the time to encourage expectations of community and respect - call out good teamwork when you see it! When time is up, give them a countdown: "30...10... 5... 4... 3... 2... 1... PENCILS DOWN, EYES UP HERE!" Wait for total silence and complete attention. Have groups volunteer some of their answers and write them on the board. When they start listing items in the background, explain that you'll be grouping them all together into "background." Assign scores to the board: "team MIT is tied with team Northeastern!"*



Thing	Changes
Cat	position
Clouds	position
Ruby	position
Dog	position
Background	nothing
Score	value

- Now take a look at that second column. We need to think about what is *changing* when we play the game. What about the Ruby? Does it get bigger? Does it change color? Does it spin around?
- The only thing that changes about the Ruby is its position! Everything else about it is the same. What about the Dog? The Cat? Fill in the rest of the second column, listing everything that changes about each character.

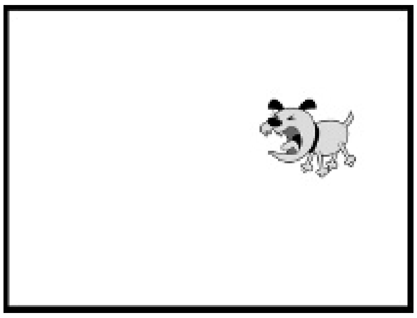
- Exercise:

  In your groups:
  Take one minute to fill in the second column, for each thing in your game.

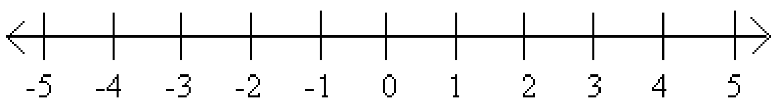
- *Again, give a countdown: "30...10... 5... 4... 3... 2... 1... PENCILS DOWN, EYES UP HERE!" Wait for total silence, and complete attention.*
- Did you notice that the Dog, Ruby, Cloud and Cat only change position, and nothing else? What about the Background - it doesn't change at all, so we can *leave that cell in table black!*. Did you notice the Score changes value? You'll want to add that to your table, too!
- *Quickly fill in the second column on the board. Assign scores.*

Coordinate Planes (Time 15 minutes)

- In our table, we've written that the *position* changes for all four characters. What is a position? Suppose you had this picture of the dog on your screen and you were talking to a friend on the phone about what it looks like. How would you tell her *exactly* where the Dog is?
  (Have students discuss.)
  Would you just say "it's kind of on the right-hand side of the screen", or would you be more specific? What if you could use a ruler?



- Just like your friend on the phone, computers need to know exactly where something is in order to draw it on the screen. To do this, they use a coordinate system.
- *(Draw a number line, with 10 tick marks). You've probably seen an image like this before.*
- There are a bunch of evenly spaced markings on the line, and each one represents a number. A number line is actually a really simple coordinate system, which lets you find points in one dimension! For example, we can take the image of the NinjaCat, stick it anywhere on the line, and now we can all agree that the cat is standing at 5, 6, or any other number on our line. Number lines can also have negative numbers, so we can put NinjaCat off the left edge of the screen at -100, or off the right edge at 800 or 900. So if we had a number line, you could tell your friend exactly how far the dog was standing to the left of the screen.
- But our game has two dimensions! How would your friend know how close to the top or bottom of the screen to draw the dog?

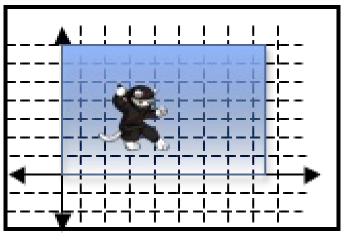


- We'll need a number line for both left-and-right measurements, and

up-and-down measurements.
(draw a second number line)

- Let's call our first line (which runs from left to right) the x-axis, and the second line (which runs up and down) the y-axis. Now we have a grid, and we can stick NinjaCat anywhere on the grid. Let's do a quick example:
- *stick NinjaCat at (4,6)*

- Exercise:



  - Here's a sample grid, with NinjaCat drawn at a certain position on that grid. What is NinjaCat's position on the x-axis? To find out, we just drop a line down from where NinjaCat is, and read the position on the number line. **NOTE:** We've chosen to drop the line down from the center of his image. Would his x-coordinate be different if we decided to always use his left side? How about his right side?
  - For the programming we'll be doing in this class, *the coordinate always means the center of the image.*
  - What is his position on the y-axis? How would you find out?
  - *If time allows, invite volunteers up to the board to try placing NinjaCat and the Ruby at different locations.*

- When we write down these coordinates, we always put the x before the y (just like in the alphabet!). Most of the time, you'll see coordinates written like this: (200, 50) meaning that the x-coordinate is 200 and the y-coordinate is 50.
- On the computer, the screen is all the parts of the grid from 0 to 640 on the x-axis, and 0 to 480 on the y-axis. *Superimpose the screen over your coordinate plane, as shown in the diagram here.*
- What is the coordinate for the lower left-hand corner of the screen? What about the lower right-hand corner? The center? (Answer: Lower left: (0,0), lower right:(640,0), center:(320,240))
- Can you think of a coordinate that puts NinjaCat on the left-hand side of the screen? The top? (Answer: Many possible answers: for example, (0,100) puts NinjaCat on the left, and (120,480) puts NinjaCat on the top.)
- Can you think of a coordinate that puts him off the screen? (Answer: Many possible answers: for example, (1000,1000). If the x coordinate is above 640 or below 0, or if the y coordinate is above 480 or below 0, NinjaCat will be off the screen.)

More dissection (Time 10 minutes)

- So let's look back at our table (Page 1). Now that we know what a position is, let's be more specific about what is changing during the game.
- Can NinjaCat move up and down in the game? Can he move left and right? So what's changing: his x-coordinate, his y-coordinate, or both? (Answer: NinjaCat can move both up and down and left and right. So both his x and y-coordinate are changing.)
- What about the clouds? Do they move up and down? Left and right? Fill in the rest of the table with your team. (Answer: The clouds only move left and right.)
- Terrific! Great job, all of you.
- Turn to Page 2 in your game planning workbook, and look at the project sheet that has a picture of the NinjaCat game you just saw.
- *Raise your hand if you can tell me what the coordinates are of the upper-left-hand corner. Take volunteers. What about the bottom-right-hand corner? What about the coordinates of the point right in the center of the screen?*

- Exercise:

In your groups:
Take a minute to find and label all of the midpoints - the points in the middle of each side of the screen: top, bottom, left and right.
(You may want to point to these points on the board).
Once again, your group only gets a point if everyone writes and labels these coordinates correctly. GO!

- *Give a countdown, demand attention, then take volunteers. Assign scores, discuss.*
- In the next round, your groups will have five minutes to fill in the coordinates for all the characters at the bottom of the page.

- Exercise:

Now fill in the coordinates for all the characters at the bottom of the page.

- *Give a countdown, demand attention, then take volunteers. Assign scores, discuss.*

Brainstorming! (Time 15 minutes)

- *Okay, teams - great job. Now it's time to just work as pairs, with you and your partner*
- It's your chance to get creative!
- On Page 3, you'll find a planning template for you to make your own game.
- Put your names at the top of your proposal.
- Just like we made a list of everything in the NinjaCat game, we're going to start with a list of everything in your games.
- To start, your game will have with four things in it:
  - A Background, such as a forest, a city, space, etc.
  - A Player, who can move when the user hits a key.
  - A Target, which flies from the right to the left, and gives the player points for hitting it.
  - A Danger, which flies from the right to the left, which the player must avoid.
- Suppose this was a sports game, like football. What might the player be?
  (A football player.)
  How about the goal?
  (A football.)
  The Danger?
  (Player from a rival team.)
- Maybe your game is about a puppy out for a walk. If the player is the puppy, what gives him points? What should he stay away from?

- Exercise:

Each pair:

Pick a setting for your game, and plan out all four characters. If you have time, you can draw a sketch for what the characters should look like.
You will have 5 minutes. GO!

- *Walk around and make sure to consult with every team. A lot of students will have trouble fitting their ideas into this format, or they'll struggle with coordinates. Be clear about what can and cannot be done! (e.g. - no 3d games, joysticks, multiplayer games, etc) There are several ways to put images into games, based on student descriptions:*
  - *The teacher can find images using an online image search, and process them using image editing software, such as Photoshop or GIMP.*
  - *The teacher can select images from the prepared image database on the Bootstrap website.*
  - *The students can use an image search website to find their own images (note: this can be very time consuming in a class setting).*

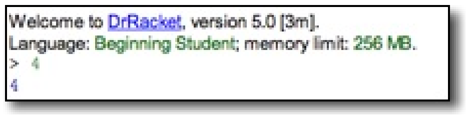
  Hint: students who have a well-worded description of their images will be happier with any images you could find them. Try searching for "Person" versus "Jogger", for example.

Intro to Racket and Numbers (Time 15 minutes)

- Okay, it's time to start learning this new programming language!
- *Have the students look at the editor on their computers. Refer to overhead projector as necessary.*
- This is a tool that allows you to write Racket programs. On your screens you'll notice two large boxes: the Definitions window at the top and the Interactions window at the bottom. For now, we're going to just focus on the Interactions window.
- The Interactions window is like scrap paper, where you can write short programs and try them out by hitting "Return." When you know what you want to keep as part of your finished product, you write that code up in the Definitions window.
- *Draw students' attention to the Language Table (see Lesson Overview for Unit 1), which currently sits empty.*
- At the very least, our language should have Numbers in it.
  On three, Let there be numbers! "1... 2... 3... Let there be numbers!"
- Add "numbers" to the Types section of the language table.
- Whenever you run a Racket program, it computes the program you've written. Since we have Numbers in our language, we should be able to write *really simple* programs, like `4`. That's right – a Number by itself is a program! Maybe not a

very exciting one, but a program nonetheless. What do you think `4` will evaluate to when we run it?

- Type in the number 4 into the Interactions window, and hit Return to see if the computer agrees with you. Congratulations: you just wrote your very first Racket program, and it came out to be the same value as what you'd expected! Try typing in other numbers and see what happens. What happens if you write a decimal?
- Try other numbers, like `1.5` (or other decimals), `10345017` (or other big numbers), and `-2` (or other negatives!).



```
Welcome to DrRacket, version 5.0 [3m].
Language: Beginning Student; memory limit: 256 MB.
> 4
4
```

- What happens when you click on a decimal that you've used? You get a new type of number, a fraction, like `3/4`.
- Every time you put a number in, without doing anything to it, you get the very same number back.
- In English we have things like nouns and verbs. What's the difference between them? In Racket, anything that is a value is like a noun. I can give Racket the number 5 and it will give it right back, unchanged. All of the numbers you've entered are examples of values.
- Racket also has things that are like verbs, called Functions. I can `throw` a marker, the same way I can `add` two numbers.
- You already wrote programs that were just values. Now you're going to write programs that apply functions to those values.

Circles of Evaluation (Time 20 minutes)

- *Draw a big circle on the board. (Tip: make this exciting for kids! Ask them to give you a drumroll, and then proudly announce what this Circle is called!)*
- This is called the Circle of Evaluation, and believe it or not it is a *fully-functioning computer*. That is, we can write programs inside the Circle, and it will evaluate them the exact same way that the computer will. If you know how to draw a circle of evaluation for a program, you'll know exactly how to write the code.
- Write 4 + 5 (jumbled, in no order in particular) in the Circle of Evaluation.

- Exercise:

  What is the value of the stuff in the circle? Let's figure out the math first. (Write the equation.) We know we're adding, so we start with the plus sign. ("+" in the middle.) Then we add 5 and 4 on either side. ("5 + 4".) Does it matter if I write (4 + 5)?

- *Let there be subtraction!*
- *Exercise:*

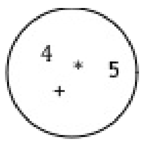
  Write a subtraction example in the Circle of Evaluation (again, jumbled), and ask students to evaluate it. Typically, students will realize that there are two possible answers! The lesson here is that the order of inputs matters. If necessary, have them type examples into the Interactions window to see that swapping the inputs gives different results!

- *Exercise:*

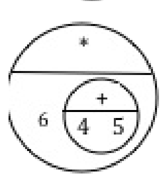
  ---

  *Write 4 + 5 * 6 in a Circle of Evaluation, and ask students what they think the value is. As with subtraction, have them discover that order of operations matters, and use the actual formulas to demonstrate: ("5 * 6 + 4", "4 * 5 + 6")*

- We need to improve our Circle of Evaluation, so we don't run into this problem. Let's use the two solutions we came up with:
  - all circles have one function
  - it matters in what order the arguments are written-We will always read and compute inputs from **left to right**.



- We'll separate the function from the inputs by drawing a line between them: the function is above the line, the inputs are below.



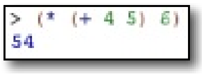
- Exercise:

How do we compute this program? Well, our new rule tells us to look at the left first: what is the value of 6? (6, of course!) Now we look at the right: that's a new circle, so we have to evaluate that first. What is the left-hand side? A 4! And the right? A 5! What are we doing to the 4 and 5? That's right, we're adding. What do we get when we add 4 and 5? (Replace rightmost circle with 9.)

- Now we have two numbers.
  What are we doing to 6 and 9? (Replace circle contents with 54.)
- We write this in Racket the same way. We always put parens before a function and after its inputs (Write the parens and the "*"). In order to apply this function, we need to compute the left side and then the right. The left side is the number 6, so we can just write that in. `(* 6 ____)`
- The right side happens to be another Circle of Evaluation (more parens), so we apply the same rules there: write the function and then look at the left and the right. These are numbers, so we can write them in directly. `(* 6 (+ 4 5))`

- Exercise:

Try writing this in the Interactions window, and hit "enter". What did you end up with? Now try writing in the complex example we used. Did you get the same answer?



```
> (* (+ 4 5) 6)
54
```

- With your partner: Try to come up with more examples - can you figure out how to subtract three numbers? (Answer: One example would be to subtract 1, 2, and 3. There are multiple ways to do this. You can do 1 - (2 - 3) which yields 0 or 3 - (2 - 1) which yields 2.)
- *Let students discuss briefly, but usher them along to the solution using nested circles. They'll have plenty of practice soon!*
- I can't put three numbers in the circle, because there's no way to tell the computer which two numbers should be subtracted first. Is it (2 - 3) - 5, or 2 - (3 - 5)?
- Well, if a complete circle evaluates to a number, and numbers go inside circles, why not try putting circles inside circles?
- *Have students practice circles - make it into a game!*
- There are several exercises provided on Page 4 of the student workbook.
- Some common bugs when writing simple arithmetic in Racket code:
  - Do your parentheses match? Be sure that you have one to open each circle and close each circle.
  - This means that you need a parenthesis before every function, and there have to be the same number of opening and closing parentheses.
  - Did you put spaces between your function and your inputs? `1 1` is not the same as `11`.
  - Remember: things like `+` and `-` are functions. The numbers that they act on, for example the `3` and `2` in `(+ 3 2)` are called inputs.

Review (Time 5 minutes)

- *Who can tell us one thing we learned today?*
- *Who saw someone else in the class do something great?*
- *Cleanup, dismissal*
- *NOTE TO INSTRUCTORS:*
  - *Make sure student names are on page 3*
  - *Take page 3 itself, or take photos of page 3, to prep game images for Unit 3.*
  - *Images should be in PNG or GIF format. Background images should be 640x480, and character images should generally be no larger than 200px in either dimension. Make sure that the character images have transparent backgrounds!*
  - *TIP: use animated GIFs for the characters - not only does the animation make the game look a lot better, but these images usually have transparent backgrounds to begin with.*
  - *For more instructions on setting up the game files, read the Coding Instructions document.*