National Coding Symposium
Quorum Intermediate Activity Lesson Plan

Compiled by Amanda Rodda

Audience:	This three lesson course is for anyone with basic computer and coding skills: ● Understands variables, conditionals, loops ● Can navigate to a website ● Can find links and buttons on websites ● Has basic typing skills If you have done any coding class in the past, you should be fine here! If you haven't, you might want to try the beginning course.

Expanded Core areas & components targeted in this lesson:

ECC Area	ECC Component

Learning targets (what do I want the students to learn?)

Objective #1	**Review basic coding concepts**
Objective #2	**Learn about actions in code and set up basic code for a game**
Objective #3	**Learn about event handling and positional audio andl use these concepts to finish your program**

Materials needed (What I need to teach the lesson?)

Teachers and students will need access to the doc files, and the Quorum website to actually code. No downloads are needed.

For this course, you will need any computer and a Quorum log in, so you can save your projects; access to the Quorum Language website, to use the online IDE, and the Google Doc lessons.

Quorum Log In

- Navigate to www.quorumlanguage.com.
- Once there, find the Login link.
    - For visual users, it is at the top of the page towards the right end of the list of links.
- That link will open a popup. Choose the Sign up for an account link.
    - Follow the directions to sign up for an account
- Once you have an account, if needed, sign in!
- You will learn how to save and load files once you make your first program.

Online IDE

An IDE is an integrated development environment. That is where you will write your code, find errors, and run your programs. There are several ways to access the Quorum online IDE. For this course, navigate to https://quorumlanguage.com/project.php. This page will allow you to use the IDE without other content getting in the way.

ECC High School Readiness Checklist sections referenced for this lesson:

Grade band:	Grades 5 - 8
ECC area: Assistive Technology	I can type at a minimum of 45 words per minute with 80% accuracy (five words per grade-- 45 by 9th, 50 by 10th, etc.). I can create, edit, save, and share a variety of formats such as Microsoft Word, Google Docs, Google Sheets, Excel, PDF, etc. I can use NVDA, JAWS, or Chromevox to complete word processing projects.

	I can use commonly used keyboard commands for reading, editing, and navigation.
Level: (Required, Novice, Accomplished or Proficient)	Required Skills

Lesson 1

Overview

In this course, you will write a hide and seek type game. The game will rely on auditory cues, and have some visual cues, so it is accessible to all players. In this lesson, you will review basic coding concepts, and make sure you understand how the Quorum language works. Make sure you read the definitions before starting the lesson, so you will understand the vocabulary used.

Computer Science Definitions

- Algorithm- written instructions to complete a task
- Assignment Symbol- the symbol used to assign a value, in Quorum it is =
- Comparison Symbols- as in math, these are symbols that compare values.
  - Equal: =
  - Greater than: >
  - Less than: <
  - Greater than or equal to: >=
  - Less than or equal to: <=
  - Not equal to: not=
- Conditional Statement- a block of code that will only run when specific conditions are met
- Input- information a computer gets from the user, devices, or other computers
- Iteration- repeating code
- Loop- a block of code that is run multiple times, in Quorum, loops end with an end
  - repeat x times- this is the format to repeat a block of code a specific number of times
  - repeat while- this block will repeat as long as the condition is true
  - repeat until- this block will end when a condition is met
- Output- information the computer gives a user, device or other computer
- Variable- a label for information used in a program
  - Boolean- a variable that only holds true or false
  - Integer- a variable that holds integer values, no decimal points
  - Number- a variable that holds number values, can have decimals
  - Text- a variable that holds text values, words or symbols, always contained in quotation marks

Other Definitions

- Adjective- describes a noun or pronoun

- Adverb- tells how something was done, usually ends in -ly
- Mad Lib- a word replacement game
- Noun- person, place, thing, or idea
- Verb- an action, something you do

Part 1: The Basics

In this lesson, you will write a Mad Lib game to review coding skills, and learn some of the Quorum syntax. The first thing you need to know is variables. Just like math class, variables hold a value. Some change, and some stay the same throughout a program. Quorum has four type of variables:

- ○ Boolean- a variable that only holds true or false
- ○ Integer- a variable that holds integer values, no decimal points
- ○ Number- a variable that holds number values, can have decimals
- ○ Text- a variable that holds text values, words or symbols, always contained in quotation marks

For this program, you will use text variables. The code below shows how to write a text variable named word, and assign it the word thanks.

```
text word = “thanks”
```

What do you notice about the sample code?

You should always declare a variable by stating the type first. Then you give it a name. The name of the variable should make sense for what you are using it for. You don’t want to name variables cat and dog, if they have nothing to do with a pet! Also notice that the variable name is lowercase. The = assigns the value. Since this is a text variable, the value thanks is in quotation marks. Now, copy the sample line, and paste it into the Quorum IDE, so retype it. Then use the Run button to run the program.

Note- Any time you copy and paste code from the lesson page to the IDE, you will have to delete and rewrite all quotation marks.

Did anything happen? Why?

Did you tell the computer to do anything with the variable?

No, you didn’t.

New statements always go on a new line, so on a new line, add this:

```
output word
```

Before running the program, what do you think will happen?

If you look in the console, you will find two lines of text. (Visually, the gray box under the buttons. Screen reader, tab past Embed.) The first says “Build Successful.” That means the computer was able to read all of the code. The second should be the word thanks. If you did not replace the quotation marks on thanks, you will have your first errors! Read them, and see if it makes sense that you should fix that word.

The last of the basics you need is user input. For our Mad Lib, we are going to use text input. The sample code below asks a question, saves it to a variable, and then outputs that variable. Enter this code and run it. Note- a popup will appear with the question. The output will always be in the console.

```
text answer = input(“What is your name?”)
output answer
```

What happened? Did the computer output a name, or the word answer? You use the variable’s name to write the program, the computer uses the value when it runs.

Part 2: Let’s DO something!

You know how to make a text variable, how to make the computer print to the screen, and how to get user input. Now let’s write a Mad Lib. To start with, you will need a few sentences to work as the story part of the Mad Lib. Below is a short introduction to Louis Braille.

Louis Braille went blind at the age of three while playing with tools in his father’s workshop. Louis first learned to read when he was ten years old. His first books were made of waxy paper with letters embossed into the pages. When Louis was 15, he took a new way to read that he learned from an Army chaplain, and improved it to become the dots we know as Braille.

Now you have a story. To make it a Mad Lib, you will take out 2 or 3 words per sentence, and replace them with a user input word. An example of words to remove, and how to write the variables and inputs is below. You are free to use these, or make your own!
Note- Using // makes the rest of the line a comment. That text will not be read by the computer when running the program. Commenting code is a great way to stay organized, and help others understand what you are doing.
Note 2- Clear your practice code, and start with a blank screen.

```
//Sentence 1
text number1 = input(“Pick a number”) //replacing three
text verb1 = input(“Pick a verb ending in -ing”) //replacing playing
text noun1 = input(“Pick a place”) //replacing workshop
//Sentence 2
text noun2 = input(“Pick a name”) //replacing Louis
text verb2 = input(Pick a verb”) //replacing read
text number2 = input(“Pick a number”) //replacing ten
text time1 = input(“Pick a timeframe, such as days, minutes, years”)
//Sentence 3
text noun3 = input(“Pick a plural noun”) //replacing books
text adjective1 = input(“Pick an adjective”) //replacing waxy
text noun4 = input(“Pick a plural noun”) //replacing pages
//Sentence 4
text verb3 = input(“Pick a past tense verb”) //replacing learned
text verb4 = input(“Pick a past tense verb”) //replacing improved
text noun5 = input(“Pick a noun”) //replacing dots
text noun6 = input(“Pick a language or way to write”) //replacing Braille
```

That was a LOT to write! Let’s take a moment to save your program. Use the Save button under the code editor. Name your project Louis Braille Mad Lib, then hit the Save Project button. DO NOT just hit enter after you name the project, it will eat your code!!

Now you will write a VERY long output statement, and use a concept called concatenation. Concatenation is when you add different parts of code together. In this case, you will add direct text to variables to produce the story. It will be much easier to copy the full story, the edit to add the concatenation. An example of the full output statement with concatenation is below.
Note- the text below has hard returns at the end of the line, as the IDE will not wrap text to a new line. This makes it difficult to read, and to find errors.

```
output “Louis Braille went blind at the age of ” +  number1 + “ while “ +
verb1 + “ with tools in his father’s “ + noun1 + “. “ + noun2 +
“ first learned to “ + verb2 + “ when he was “ + number2 + “ “ +
time1 + ” old. His first “ + noun3 + “ were made of “ + adjective1 +
“ paper with letters embossed into the “ + noun4 +
“. When Louis was 15, he took a new way to read that he ” +
 verb3 + “ from an Army chaplain, and “ + verb4 + “ it to become the “ +
 noun5 + “ we know as “ + noun6 + “.”
```

Run your code and see how it goes! Make sure to save your final changes, so you can share your code with others. In lesson 3, you will learn how to load a saved project.

In the completed example below, there is a second output that tells the correct story. You can add this to your code, or just leave it as the Mad Lib.

You can go on to Lesson 2.

Want to see the project all in one place, and run it?

Lesson 2

Overview

You will start programming a hide and seek type game. In this lesson you will learn about actions, and set up the static elements of the game. You will need this game engine template. Once you open the template, save it as Hide and Seek Game. The sample project will use a starry space background, and a space shuttle looking for the international space station. These are images available to the online IDE. You also have a reference list of all assets available, to customize your game.

Definitions

- Action- behaviors a program can take, reusable blocks of code that may or maynot hold parameters. In Quorum, actions end with an end.
- API- Application Program Interface- specifications on how actions in a library behave and can be used
- Library- a group of actions that can be used to write new programs
- Parameter- a variable in the definition of an action, a placeholder for values passed through the action

Part 1: The Basics

The template starts with a use statement to import the game library. Libraries are collections of actions that can be used to create programs. You will need other libraries, specifically the ones that handle images and sounds. Add this code to your project on the lines under the game line.

```
use Libraries.Sound.Audio
use Libraries.Game.Graphics.Drawable
```

The rest of the template is several actions. Actions are blocks of code that can be used and reused in a program. The first action is Main, Quorum always starts with this action. You will see in the action, another action. StartGame() is an action written into the language that tells the program the order to call actions in. It instructs the program to read CreateGame(), and complete anything there, then go to Update(). Update is then called repeatedly, the speed depends on the machine the program is running on. Faster machines will load the Update action more often than slower ones. Think of computer frame rates. Update runs once a frame.

Part 2: Let's DO Something!

Now you will create Drawable objects, that is what images are called in the game library, and Audio objects. Objects are similar to primitive variables, the difference is objects connect to a library, and can have many different actions called on them. You will create the objects for your project as global objects. That means they are created in class Main, but not in any action. This allows all actions to use the object. To create an object, you start by stating the class or library it comes from. In this case, you will use Drawable and Audio. Notice that both words start with a capital letter. Then, you will name the object, just like naming a variable. The sample code below should be placed after the line class Main is Game.

```
Audio targetBeep
Audio success
Drawable background
Drawable target
Drawable player
```

If you run your code, nothing exciting should happen. Next you need to load the assets to the objects. For both audio and drawable objects, you will used the Load action, and put the file path in the parameters. In Quorum, a colon is used to call an action on an object. These are static elements, so the code will be in the CreateGame action.

```
//Load audio
targetBeep:Load("media/Bing.ogg")
success:Load("media/Firework.ogg")

//Load drawables
background:Load("media/astro/galexy.png")
target:Load("media/epiq/Shuttle.png")
player:Load("media/epiq/SpaceStation.png")
```

You will now add the drawables. When adding images, remember that the computer will add from back to front. If you want a background, add it first. Add any character or obstacles next. In the Quorum game engine, images are placed at (0, 0), which is the bottom left corner of the screen. The sample code uses SetPosition and ScaleFromCenter to place the target and player, and change the size of the image. The parameter for scaling is by percentage as a decimal. This code is added in CreateGame directly below the load statements.

```
//Set position and size of drawables
background:ScaleFromCenter(1.2)
```

```
target:ScaleFromCenter(0.3)
player:ScaleFromCenter(0.2)
target:SetPosition(300, 200)
player:SetPosition(35, 35)

//Add drawables
Add(background)
Add(target)
Add(player)
```

You will set up the audio aspects in the next lesson. If you would like to hear the sounds chosen, use the sample code below. You will delete these lines before the next lesson, but they do show how to play a sound. Place this code under the add statements, still in CreateGame.

```
//Temporary code to here audio files
targetBeep:PlayUntilDone()
success:PlayUntilDone()
```

You now have all the static elements set for your game. In the next lesson, you will learn how to add sounds to objects, and how to make the player move.

You can go on to Lesson 3.

Overview

In this lesson, you will learn about event handling and positional audio. You will use these concepts to finish your program. You will also be given code to manage the collision of the shuttle with the space station using the distance and volume variables you create.

Definitions

- Collision- when two graphics' bounding boxes, outer borders, touch
- Event Handling- a method for using input; sets blocks of code, actions, to run when an event occurs
- Inheritance- allows a class or object to uses all the actions of another class or object
- Positional Audio- sound set to a position in 3D space; uses the 3D coordinate system (x, y, z). Quorum's 3D configuration has the x-axis horizontal on the screen, y-axis vertical, and the z-axis through the screen, seeming to be in front of or behind the speakers

Lesson 3

Overview

In this lesson, you will learn about event handling and positional audio. You will use these concepts to finish your program. You will also be given code to manage the collision of the shuttle with the space station using the distance and volume variables you create.

Definitions

- Collision- when two graphics' bounding boxes, outer borders, touch
- Event Handling- a method for using input; sets blocks of code, actions, to run when an event occurs
- Inheritance- allows a class or object to uses all the actions of another class or object
- Positional Audio- sound set to a position in 3D space; uses the 3D coordinate system (x, y, z). Quorum's 3D configuration has the x-axis horizontal on the screen, y-axis vertical, and the z-axis through the screen, seeming to be in front of or behind the speakers

Part 1: The Basics

For this lesson, you will need to load your Hide and Seek project. Navigate to the My Projects link. Visually, this is at the top of the screen near the Log Out link. On this screen you will find a list of saved projects. Checking the public box allows you to share a link to your code. The share button will give you the link. You don't need that right now. Navigate to the Load button. This will take you to an IDE with your code. You will also notice an Embed button. If you have your own website, you can embed a Quorum IDE into it, with the code to your project, allowing others to run and play your game

The more complicated concept for this lesson is event handling. You will use the KeyboardListener, a KeyboardEvent libraries to write an action to run when the specific keys are pressed. That part will be explained in Part 2 of the lesson. For now, add the following libraries to the top of your project.

```
use Libraries.Interface.Events.KeyboardListener
use Libraries.Interface.Events.KeyboardEvent
```

You also need to add to the class Main line. It already inherits from the Game library, allowing all Game actions to run, now you will add KeyboardListener, to allow for listener actions. Change the class Main line to the code below.

```
class Main is Game, KeyboardListener
```

You also need to add the keyboard listener to the CreateGame action. This is its own action, and takes a kind of funny parameter. You will put me in the parenthesis. That tells the computer that the current class is what is being made the listener. Delete the play commands you put in yesterday, and put this code at the end of CreateGame.

```
AddKeyboardListener(me)
```

For the positional audio, you will need to add the Vector2 library, and two Vector2 objects. In this case, the Vector2 objects will hold the x and y coordinates of the player and target to check the distance between them. Place the use statement with the other libraries at the top of your code, and create the Vector2 object under the last Drawable.

```
use Libraries.Compute.Vector2

Vector2 targetPoint
Vector2 playerPoint
```

Part 2: Finish Hide and Seek

You are now ready to add movement to the game. First, you will add a form of positional audio to the space station compared to the shuttle, so a player with no vision can use it to find the space station. You will use a couple of actions to find the distance from the player to the target. Then you will write a conditional to change the volume of the sound as the player gets closer. The code below goes in the Update action because you want the positions checked often, to make the sound's volume changes with the movement.

```
playerPoint:Set(player:GetX(), player:GetY())
targetPoint:Set(target:GetX(), target:GetY())
number distance = playerPoint:Distance(targetPoint)
number maxDistance = 400
number volume = 1 - (distance / maxDistance)
if volume < 0
  volume = 0
end
targetBeep:SetVolume(volume)
```

Your game should now beep fairly quietly when you run it. Now, you need to add the action that will control movement. The sample code below shows how to make the player move up and right. You need to add elseif statements to make it move down and left. The sample code starts using WASD, a common set of keys for game movement. If you want to use arrow keys, use UP, DOWN, LEFT, and RIGHT. Make sure they are in all caps. This code goes after action Main's end, and before the class main end. Changing the amount added or subtracted will change the player's speed.

```
action PressedKey(KeyboardEvent press)
        if press:keyCode = press:W
                player:SetPosition(player:GetX(), player:GetY()+5)
        elseif press:keyCode = press:D
                player:SetPosition(player:GetX()+5, player:GetY())
        end //end conditional
end //end action PressedKey
```

Now, to make it more fun, you will hide the space station until the shuttle gets closer. This will make the game more like hide and seek, and not just "move the shuttle to the space station." The hide line goes in CreateGame. The show conditional goes in Update. The conditional uses the distance variable already set up to change the volume. You don't have to make a new one.

```
target:Hide()
```

```
if distance <= 75
        target:Show()
end //ends target show conditional
```

You now have a working game! There are a few choices for ending the game. You can add a conditional that checks the volume = 1, and success:PlayUntilDone(). That would play the fireworks sound when the player is that close to the target. You can also add targetBeep:Stop() to turn off the beep at that time, and even a say statement to tell the player they have won.
Note- If you copy and paste the sample code, you will have to delete and retype the quotation marks.

```
//add this code after volume = 0
elseif volume = 1
        targetBeep:Stop()
        success:PlayUntilDone()
        say "Congratulations! You won!"
//make sure the above code goes between volume = 0 and the end
```

Congratulations! You have completed the APH Coding Symposium Quorum Language Intermediate Course!

Want to see the final project and run it?

Want to change up the images or sounds? Check out this reference sheet!