

Iteration and Loops: Guessing Game - One Player

goals

- Learn to use *while* loops in a program
- Learn to increment a count
- Apply coding fundamentals to create algorithms
- Pair program to develop an app for creative expression



description of app

Add a random number generator to modify the app from the previous activity from a two player game to a one player game.

Essential Questions

1. What are the advantages and benefits of using loops in an algorithm?
2. What are the advantages and challenges of pair programming?
3. What are all the different ways iteration plays a role in a program and in an app that is created for others?

Essential Concepts

- Pair Programming
- While Loops and Incrementing Counts

Resources

[one-player Guessing Game app icon](#)

Pair Programming

What if you were instructed, “Before the end of this period, write a program that will display all the integers from 1 to 2000 on the screen and have a computer randomly select one of those numbers.”

Would you have time to write each individual number in the program? Or would you need to find a way to get the computer to generate each number for you in a matter of seconds? Using incrementing variables, a program can take a starting variable of 0 and add to that variable, printing each number, to do all the work for you.

You will not have to do this type of programming by yourself. You will have a partner to work through this challenge to expand the Guessing Game together. So far, you have worked independently, relying on those around you for help when you need it. But the programs are getting more complex, so now you will work with someone else to help speed up and streamline the programming process.

Review the pair programming content from the Collaboration Strategies page:

[Collaboration Strategies: Pair Programming](#)

PAIR PROGRAMMING

The Driver and Navigator should always **communicate** and **collaborate** with each other!

DRIVER

- Writes and modifies the code.
- Controls the computer or tablet.
- Transfers the code to any external hardware or devices.

NAVIGATOR

- Keeps hands off the computer.
- Reviews the code, identifies potential errors, and proposes ideas and suggestions.
- Connects wires to any external hardware or devices.



PLTW DEVELOPER'S JOURNAL After you review the information about pair programming, complete the following:

1. Summarize what pair programming is.
2. Describe what characteristics you are looking for in a partner that you can model.



Design Overview: Guessing Game Iterating on the Previous Activity

User Story

In the last activity, you laid the groundwork for a two-player game for students to play on the bus. You are going to expand the Guessing Game app from Activity 1.1.4 to allow one person to play by themselves. You will do this by making a loop with local variables to generate a random number based on how many integer place values the player wants to guess from. To add a challenge, you will also add a countdown timer.


Design Update

Add the following features to the Guessing Game:

- ☐ The first feature will allow a user to play the game without a partner entering a new number or guessing at the pre-programmed value of 55. Instead, the program will generate a random number for the user to guess.
- ☐ The second feature is the ability for the user to select how big of a number, the number of digits the number will have that they want to guess. They will be able to enter “1” and guess at a number between 0 to 9, or enter “4” and guess at a number between 0000 to 9999.
- ☐ The third feature is a countdown clock that will allow users to have an additional challenge of trying to guess the number before time runs out.

Updated Backlog Breakdown

- ☐ The app needs an input for the user to enter the number of place values they want to guess at:
 - Entering 1 would provide 0–9
 - Entering 2 would provide 0–99
- ☐ The app needs an algorithm that will generate a random number 0–9.
- ☐ The app needs to loop the algorithm for generating a random number the number of times the user enters.

-  The app needs a countdown clock that is visible to the user.
-

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

1. Navigate to MIT App Inventor and log in.
2. Open the 2 player Guessing Game that you previously created.
 - Do a save as, and save the app with a new name to show it is being iterated on to become a 1 player game.
3. Form pairs as directed by your teacher.
 - Take a moment to determine whose program from the last activity you will use during this activity.
 - Providing the app will designate that partner as the navigator, and the other partner the driver, to start this activity.
 - Establish how you will share files. Having this conversation now will help you know how to share the file at the end of class.

Reminder: At the end of each class, be sure to share the file with your partner so that if you or your partner is absent for the next class, neither of you will get too far behind.

4. Add a *TestingLabel* to test whether the program you are developing works, as you develop it.

During app development, you will use this label to see the generated number and confirm that the app is functioning properly. After the app is working properly, you can remove this label or set it to not be visible unless the game ends.

5. Add a *Textbox*.
 - Rename it to *NumIntegerInput*.
 - To set the *Properties* of the *text box* to accept only numbers, select the box next to **NumbersOnly**. Because it will only allow a user to enter numbers, this setting prevents you from having to create a nested conditional to check for only numbers first.
6. Add a *Button* and rename it *NumIntegerButton*.
7. Design your user interface to help a user know what to do. Consider whether using a *Horizontal Arrangement* will help keep the label and text box together in the user's mind.

Using Loops

In the last activity, you incremented the number of guesses value stored in variables. With every guess, the guess variable was decreased by one. A **loop** automates a process that occurs multiple times based on a starting input and an exit condition. Another word for a loop is an **iteration**. Each time the loop executes is called an iteration. You have been doing an iterative process by going

through your code and modifying it for a specific purpose each time. You may have also seen iteration of variables in games you have played.

Games that have a life counter move through the game until you lose one of your game lives. Then, the game loops through the next segment or level of the game until your life counter runs out or you exit the loop by getting to the end of that segment. Each time through the loop, the iteration makes the number of lives variable have one less life until the counter gets to 0. Then, the game ends.

In this activity, you are going to learn about loops. Loops are used to repeat steps or procedures continually until an end condition is met. You may be familiar with loops from outside of computer science, such as in sheet music.

Music uses repeat features and symbols for sections like the chorus to save the amount of space the music takes up. The sheet music specifies a certain number of times to repeat a section, and when that number is met, you move on to the next part of the music.

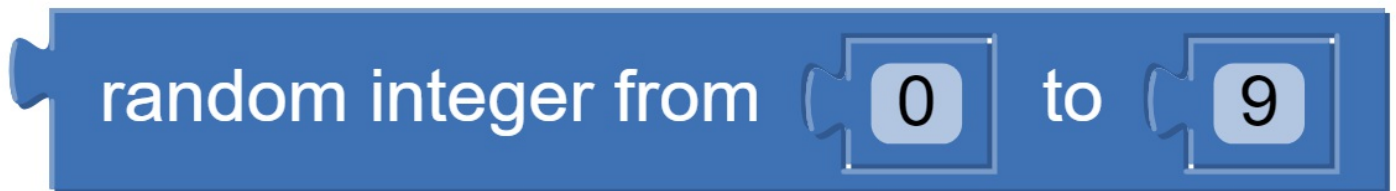
Implementing a Loop

Playing a guessing game that never changes what you guess is not very fun. You need to develop an algorithm that will randomly generate a new number each time the game is played. To make the app more marketable, it is helpful to give the user some control over the level of challenge they want.

Small children may only want to guess a number from 0 to 9, while an older child may want to guess from 0 to 999. Some people may want to really challenge themselves by guessing from 0 to 99,999.

To make this possible, you can set up an algorithm that will loop as many times as it takes to provide the user with the number of place values that meets their preferred level of challenge.

In the *Math* drawer, there is a block that can help generate a random number:



The game needs to allow the user to enter what the number range should be. The ones, tens, and hundreds places can each hold the same range of numbers from 0 to 9. The goal is for the program to randomly generate each one of these places individually for as many place values as the user indicates.

Using these *random integer* blocks, it is possible to use a loop to do the math for us instead of entering all the possibilities that the user may want.

Using Variables in Loops

The user wants to generate small or large numbers based on their preference. Using a loop to call the *random integer* block will allow large or small random numbers to be generated based on how many times the loop iterates. If the user wants to guess up to 9999, they would enter “4” in the

NumIntegerInput *TextBox* because they want 4 place values. The value “4” is stored in the variable that triggers when the loop needs to stop.

Iteration of generating a random number between 0–9 and concatenating each time through a loop

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.



PLTW DEVELOPER'S JOURNAL

1. First loop iteration will generate a number 0–9
Might look like a 3
2. Second loop iteration will generate a number 0–9 and concatenate it onto the first one
Might look like a 37
3. Third loop iteration will generate a number 0–9 and concatenate it onto the first two
Might look like a 371

What might the fourth, fifth, and sixth iterations through the loop look like?

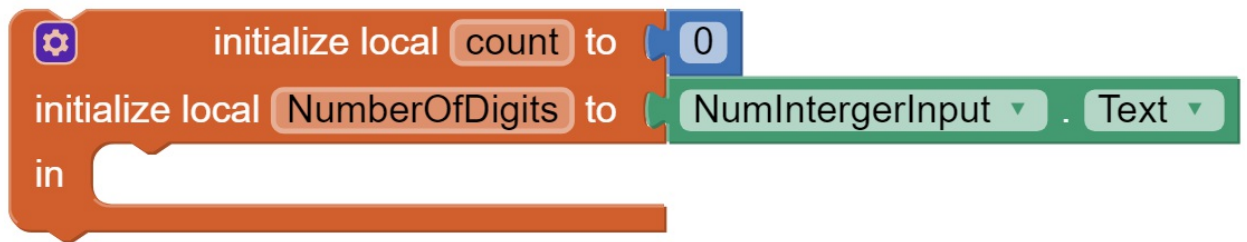
Important: An important note to take here is that the computer moves through the loop so fast that, to the user, a four-digit number was generated instantly altogether, not single numbers at a time creating a four digit number.

Setting Up the Local Variable

Using variables in the loop will help track how many times the loop has iterated. Much like in the example above, having a count for the loop number allows you to know how many numbers you should see generated. It is important that the values for both the loop number and the desired number of loops are stored in variables for the computer to increment and compare.

Will the count variable and the number length variable be needed by the rest of the app or only the loop? The answer to this question will let you know whether you should use a local or global variable.

8. Talk with your partner and set up the variables. Make sure your variable names are specific, can be quickly differentiated, and are as short as possible.
 - Loop count variable - What number should this variable be initialized to to give an accurate starting count each time the loop starts?
 - Desired number length variable – Should be determined by the *NumIntegerInput.Text*.



PLTW DEVELOPER'S JOURNAL Within what event handler do you need the computer to generate a new random number? Hint: When will a player need a new random number generated?

- Discuss with your partner and add the *control loop* block to that event handler.

While Loop to Generate a Random Number

You have the variables set up, but there is nothing that will make a loop to increment those variables.

- To make a loop inside the *local variable*, you need to add a control block, specifically, a *while test do loop*.

As the name suggests, while certain conditions are true, this block will do whatever is in the loop. The *while test* evaluates whatever is plugged into it to determine whether the Boolean result is true or false.



- If the condition is *false*, then the program does nothing with the *do* part, and continues to any code after the *while_test_do_loop*.
- If the condition is *true*, then the program moves into the *do* portion and performs whatever you put there.

People use these types of loops without thinking about it sometimes, such as when driving:



Driving Loop Example

While you are on your way to school, you will drive through yellow and green lights.
While the lights are red or you arrive at school, you will not will continue to drive.

When the light changes from red to green, you will continue your drive.

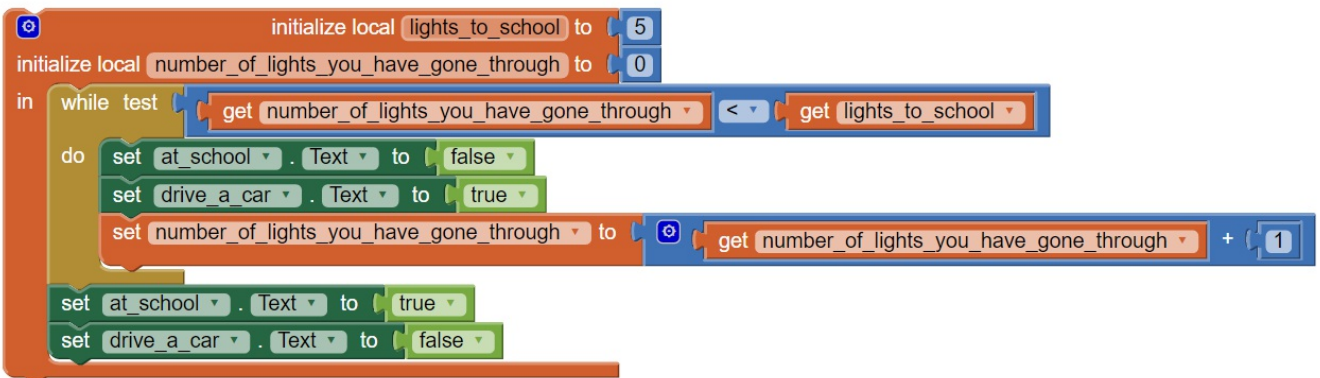
This driving loop expresses that *WHILE* the light is green *OR* the light is yellow, *DO* set your car to drive as true (drive your car). *While* the loop is not true—the light is *NOT* green *OR* yellow—set the car driving to *False* (do not drive).

A *while* loop only loops as long as it receives a “true” value. The loop stops on reds or when you arrive at school.

While Loops Inside a Local Variable

Combining a loop into a local variable helps control the number of times the loop iterates. You may know how many lights you must go through before getting to school. Every time you go through a light, you know you are one light closer to being done with your drive. Until you have gone through all the lights, you know to keep driving, because you are not at school. Once you have gone through all the lights, you know you are at (or close to being at) school, so you are done driving.

11. Talk through this example in natural language to plan what it would look like in a block-based program.



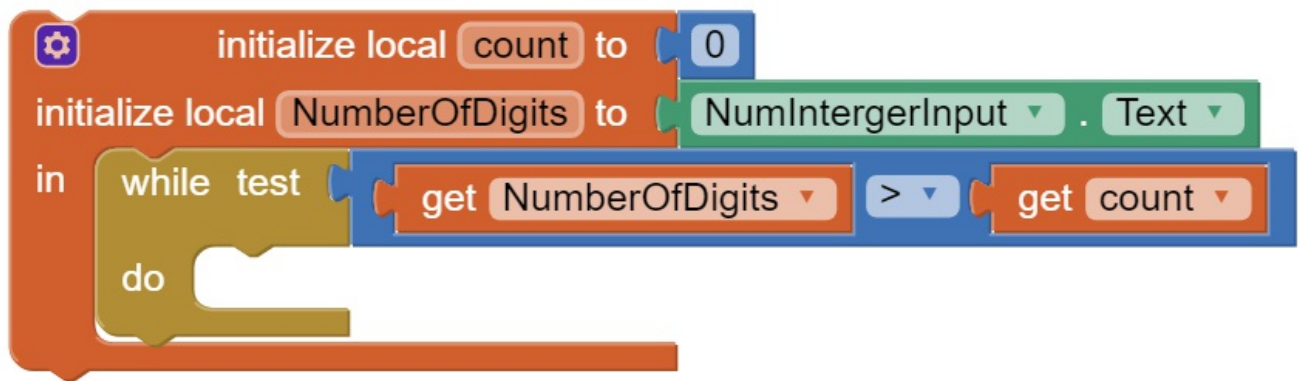
Driving Example Using a Local Variable

In block-based programming, it is possible to set up this example of counting lights on the way to school in a loop using local variables.

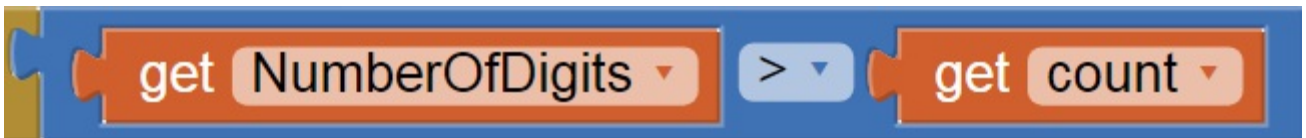
12. Set up a *while* loop inside your local variable. The *while* loop will test whether the loop count is less than the number of digits that the user wanted. As long as this is true, it will keep generating a number and incrementing the loop count. Once the number of loops is no longer less than the number of digits the user wants, the loop will stop.

To set up the test portion, you need to focus on whether or not the count variable is equal to the *NumberOfDigits*. You can use arithmetic operators to compare the value of the two variables. Remember, you can hover over the variables to pull out the *get variable* block.

13. Add a *while test do* loop block inside the *local variable* with the count and *NumberOfDigits*.



Setting up the test input of the loop using arithmetic operators may look something like this:



PLTW DEVELOPER'S JOURNAL Could the test part of the loop also look like this? Why or why not?



Counted Loops Through Incrementing Variables

Each time through the loop, the loop adds one place value to the value that is stored in the count variable until the count number (the number of place values) is equal to the number the user put into the *NumIntegerInput*, which is stored in the *NumberOfDigits* variable. When those two numbers—the number of place values the user wants and the number of place values created—are equal, the loop will stop.

You need to set up a counter inside the loop so that eventually the counter will be the same as what the user wanted, and the loop will stop.

14. Inside the *do* portion of the loop, use arithmetic operators to set the local count variable to be the *local count* variable plus 1.

Each iteration will add 1 to the count, and the loop will eventually meet the end criteria to exit the loop. Now that the loop is being counted, an action needs to occur each time, rather than just counting the loop iterations.



15. Set the *TestingLabel.Text* to concatenate with a random integer from 0 to 9. This way, the first time through, the loop will generate a single digit number. The second time through the loop, it will generate a single digit number that it will add after the first loop's number in the *TestingLabel.Text*, making a two-digit-long number.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

16. Reflect on the functionality of the game. If a user wants a six-digit number, and enters "6" inside the *TextBox*:



PLTW DEVELOPER'S JOURNAL

1. How many iterations of the loop will the loop go through before it stops?
 2. How many digits will the number produced by the loop have at the end?
17. Test, debug, and adjust.
 - ☐ A random number appears in the label each time you trigger the event handler.
 - ☐ You see the same number of integers compared to how many you entered as a user.
 - ☐ Touch the *PlayButton*. (Click a few times then move on in the activity.)

Important: Remember to save your project with a new name before you continue.

Resetting the App Play

18. Change roles as navigator and driver.
19. Talk with your partner to determine what you might need to update for the end-of-game conditions.

You may have noticed that the loop keeps adding numbers, but it does not clear them when you try to get a new number. Every time the user clicks a button to play a new game, the program needs to clear the *TestingLabel.Text* by setting it to a blank string.



- Be careful about placement, because putting the *set* blocks in the wrong spot could clear the new number.
- Talk with your partner before adding the *set to blank* string blocks.

20. Test, debug, and adjust.

- ☐ All previously existing functions still work.
- ☐ A random number appears in the label each time you trigger the event handler.
- ☐ You see the same number of integers compared to how many you entered as a user.
- ☐ Test the app by continuing to push the *PlayButton*.

21. If the new feature works, it becomes important to test previous parts of the app to make sure they still work as expected before you go too far.

Connecting New Features to Previous Features

Think back to the last activity and review your existing code. Where does the game get the winning number? The program compares the number the user entered for their guess to the *global number* variable. Have you set that global variable to the new randomly generated number that is stored in the *TestingLabel.Text*?

To connect the new feature to the existing features, you need to modify the global variable to get the number from the *TestingLabel.Text* where the local variable outputs the generated number.



22. Inside the *PlayButton* event handler, add the blocks that will set global number to *TestingLabel.Text*.

- Talk with your partner about where you should add that code.
- Add the code and test your app.

23. If everything is working, you now need to hide the *TestingLabel* from users. Decide how you and your partner would like to accomplish this:

- Option 1: Set the component to “invisible” in the *Designer* Properties.
- Option 2: Set the visible blocks to “false” for the *TestingLabel* in the *Blocks* view.

24. To make sure your app is working, play the game with your partner.

Using a Clock to Countdown

After playing the game, you may realize that it lacks some challenge. The great thing about incremental building and testing is that you can always expand what you have done.

To increase the challenge of the game, and lower the possibility for getting outside help if playing

with two players, you are going to add a timer that counts down.

To guide the process of adding a countdown clock, review what you have already created in your app. To create the clock, you will rely on your knowledge of variables and loops.

25. In the *Designer* view, add a *Clock* component from the *Sensors* drawer.
26. Near the *LowerHigherLabel* and *GuessesLeftLabel*, add a *CountDownLabel* to the *Horizontal Arrangement*.

Important: In a previous activity where we reviewed variables, there was a discussion about when to use global variables and when to use local variables. Keep that discussion in mind as you think about setting up the clock variables. Would a local or a global variable be better?

27. In the *Blocks* view, add a global *Initialize Timer* set to “20”.
28. Drag out the *When Clock1.Timer* event handler to the viewer.
29. Add an *if-then-else* control block to the event handler.

What you want to happen in the conditional is an incremental decrease in the timer. However, unlike the other parts of this app, the timer will trigger every second; every second it will automatically increment with only input from the timer, not the user:

```
If the global timer = 0 (use logical operators)
  Then
    Users sees that they lost.
    The clock stops counting.
  Else
    The timer variable increments down by 1.
    The user sees the new time.;
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

30. To set up the *if* portion of the event, use logical operators to compare whether the *global timer* is equal to 0.
31. Set up the *then* part of the conditional.
 - In the *WinLoseLabel*, the user should see a message that they lost.
 - The clock *TimerEnabled* property should be set to false so it stops counting.
32. Use what you did to increment the *guesses count* inside the *Clock.Timer* in a conditional statement to enable a clock countdown. Look at the *Clock.Timer* event handler and talk through with your partner:
 - What should the variables read instead of *global guesses*?

- What label will you modify?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

33. Add to the *If win* part of the *GuessButton* event handler conditional to set the *clock enabled* to “false”, so the time will stop and let the user know how fast they were.
34. Add logic blocks to the *Else If lost* part of the conditional (the *no more guesses* part of the *GuessButton* event handler conditional) to set the *clock enabled* to “false” so the clock stops.
35. Add to the *PlayButton* event handler to set the *clock enabled* to “true”, so when a user plays alone, they can play against the clock.
36. Add to the *EnterNumButton* event handler to set the *clock enabled* to “true” as well.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

37. Test, debug, and adjust.
 - ☐ All previous parts of the app still function correctly.
 - ☐ Restarting the game starts the countdown over at 20.
38. When your code is working, let your teacher know.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Conclusion

1. What are two distinct difficulties or opportunities you encountered and how they were resolved or incorporated?
2. Describe the incremental and iterative development process of your program.
3. How did you interpret and respond to the essential questions? Capture your thoughts for future conversations.