

Unit III: Advanced Techniques

Chapter 16: Random Numbers

In this lesson you will learn more about random variables. We can use random numbers in many ways. One thing we can do with random numbers is to flip a coin, roll dice, or select a card from a deck. Those are things that we think of as random.

We can also use random numbers to make movement seem more natural. When a person jumps up and down, they don't jump exactly the same amount each time, so we could have them jump a random amount within a range.

In this chapter, you will learn how to use random numbers in Alice programs to create actions and outcomes that are less predictable. You will explore how random numbers can be used in games to vary events, choices, and results, making each experience different. You will also learn how randomness can make the movements and behaviors of 3D characters appear more varied and natural, helping you create more engaging animations and interactive programs.

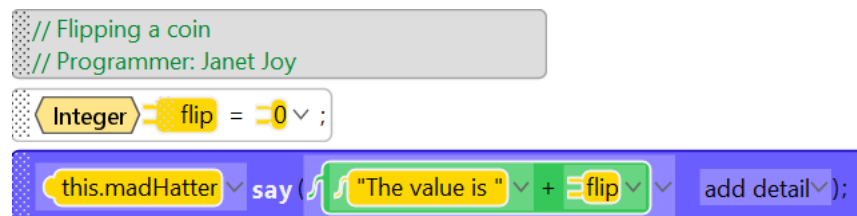
Objectives

Write code to generate random numbers in 2 different ways:

1. Use random values to create games such as flipping a coin and rolling dice;
2. Use random values in your animation so that characters move more naturally. (For example, a fish swimming back and forth doesn't move the same amount each time.)

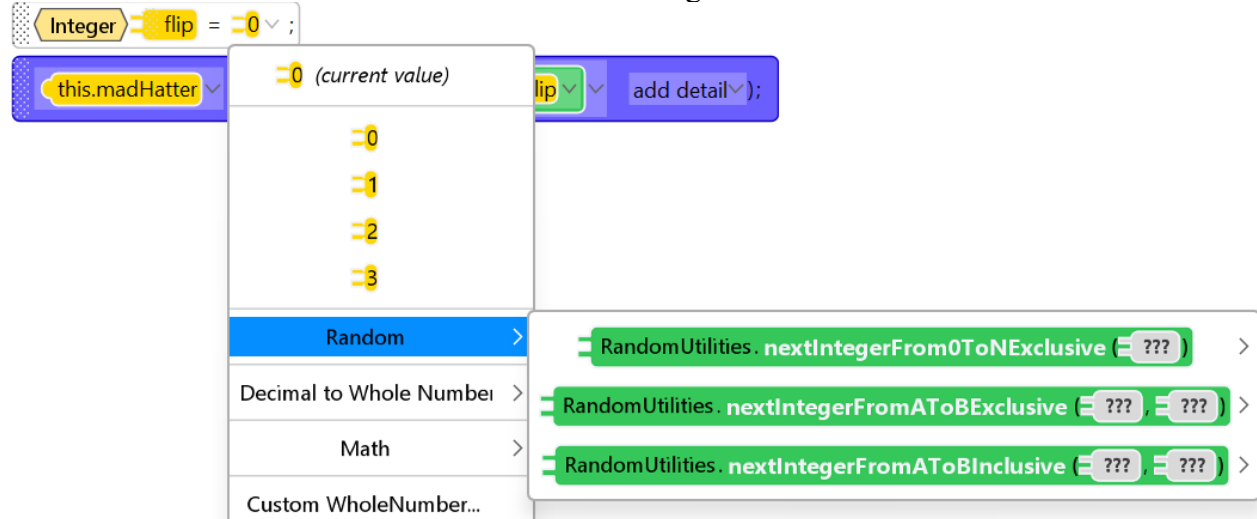
Flip a Coin

When we flip a coin, there are two possible outcomes: it lands with either the head or tail facing up. If it is a fair coin (not weighted, or bent, etc.), it will land heads about half the time and tails half the time. In other words, the chances of it coming up heads is a 50% chance. If we toss the coin 100 times, it should be heads about 50 times. Not always exactly half the time, that's what makes it random. **Each time you flip the coin it has a 50% chance of landing heads, no matter what it has done previously.**

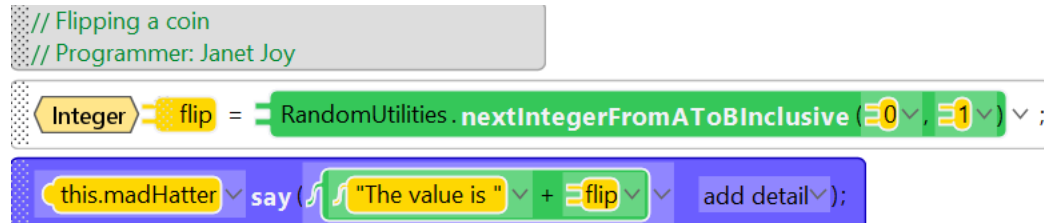


We start by declaring an integer variable **flip** with an initial value of 0. Then the madHatter says the value.

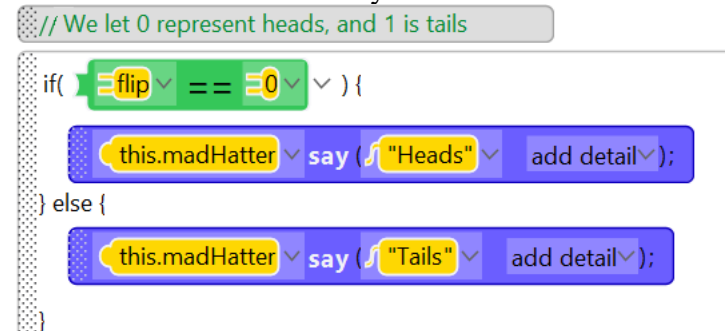
Next, we select the 0, and replace it with a random value from 0 to 1 inclusive. After making the selection show below we are asked to select two integer values: we select 0 and 1.



The code now looks like this:



When we run the program the value of flip will be either 0 or 1 and value is displayed. We can use an if block to say the words “Heads” or “Tails” :



Here, we decided that 0 would represent heads and 1 would represent tails.

Next we will let the user pick heads or tails, then flip the coin and tell the user if they won or lost.

Here is a truth table showing all the possible outcomes. The user should win about half the time.

User	Flip	Result
heads	heads	User wins
heads	tails	User loses
tails	heads	User loses
tails	tails	User wins

Experiment: try to implement this before turning the page.

One solution is shown below:

```
public void myFirstMethod() {
    // Flipping a coin
    // Programmer : Janet Joy
    Integer choice = this.madHatter.getIntegerFromUser( "Pick 0 for heads, 1
for tails." );
    Integer flip = RandomUtilities.nextIntegerFromAToBInclusive( 0, 1 );
    this.madHatter.say( "The value is " + flip );
    // We let 0 represent heads, and 1 is tails
    if( flip == 0 ) {
        this.madHatter.say( "Heads" );
    } else {
        this.madHatter.say( "Tails" );
    }
    if( choice == flip ) {
        this.madHatter.say( "You win" );
    } else {
        this.madHatter.say( "You lose" );
    }
}
```

We made this a bit simpler by asking the user to select an integer 0 or 1. Of course, the user could pick any number, but if he picks anything other than 0 or 1, he loses, because the value of flip is either 0 or 1.

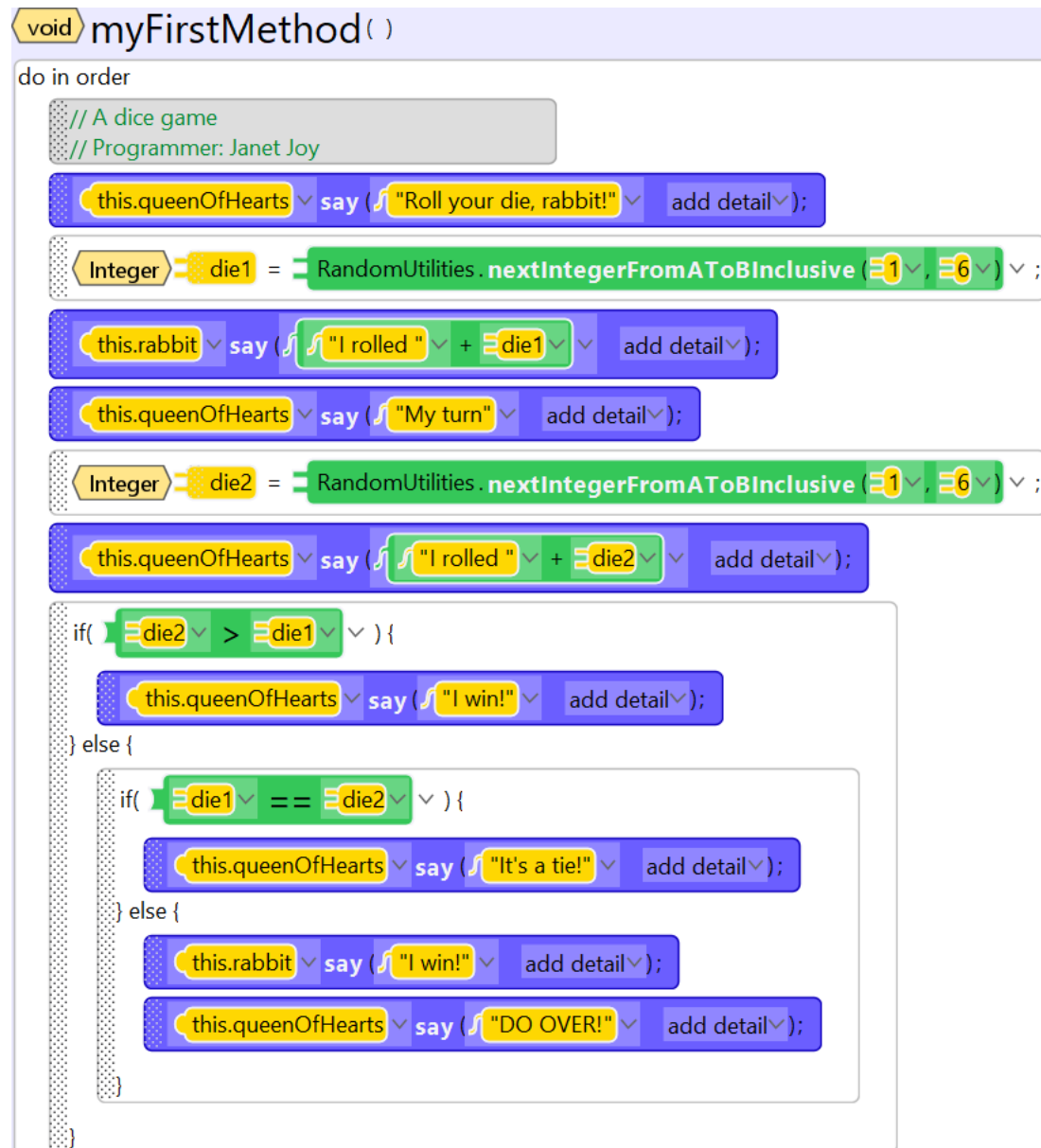
Experiment: Loop to flip the coin 100 times. Count how many times it comes up heads. Display the results.

Rolling Dice

When we roll one die, there are six possible outcomes: it lands with the number of dots on the top side is 1, 2, 3, 4, 5, or 6. If we roll two dice, there are 36 possible outcomes: Die1 is either 1, 2, 3, 4, 5, or 6; and die2 is also six possible outcomes. You can create the truth table for this. If you want to consider just the total, then there are 11 possible totals (2 to 12, inclusive), but some totals appear more often. There is just one way to roll 2: both dice roll 1. How many ways are there to roll a total of 6?

We can make a game where two characters each roll a die. The winner is the one who rolls the highest number. If the two numbers are the same, it is a tie.

Experiment: Try to write this on your own before you look at one solution.



Of course, the queen won't admit she lost!

Jumping

Random numbers can be used to make animation more realistic. Unless we are marching or dancing, our steps are not exactly the same amount, our hands don't move the same amount each time we clap, we don't jump the same amount each time we jump. Adding some random values for movement can make the animation look more natural.

We are going to make the rabbit jump up and down within a range of values to look more natural. The amounts to turn, roll, or move are always double, so we will declare a double **amountJump**, then use a loop to make him jump up and down several times.

To start, we will just have him move up and down the same amount and jump three times.

```
void myFirstMethod ( )
do in order
  // Bunny jumping random amounts
  // Programmer: Janet Joy
  Double amountJump = 0.5 ;
  // Bunny moves up then down the same amount
  for( Integer indexA = 0; indexA < 3; indexA ++ ) {
    this.bunny move ( MoveDirection.UP, amountJump add detail );
    this.bunny move ( MoveDirection.DOWN, amountJump add detail );
  }
```

Next we are going to give amountJump a new random value inside the loop. *If we do it **outside** the loop, he will jump the same random amount three times.*

```
void myFirstMethod ( )
do in order
  // Bunny jumping random amounts
  // Programmer: Janet Joy
  Double amountJump = 0.5 ;
  // Bunny moves up then down the same amount
  for( Integer indexA = 0; indexA < 5; indexA ++ ) {
    amountJump = RandomUtilities.nextDoubleInRange ( 0.3, 0.7 );
    this.textModel setValue ( "Jump " + amountJump );
    this.bunny move ( MoveDirection.UP, amountJump add detail );
    this.bunny move ( MoveDirection.DOWN, amountJump add detail );
  }
```

We added a textModel so that we could display the amount to jump each time. The value 0.5 seemed like a good place to start, so we used a minimum value a bit less than that and a maximum value a bit more. We want the difference to be noticeable so we used values from 0.3 to 0.5.

Experiment: Make the number of times he jumps a random number.

A Guessing Game

We use random numbers to create a guessing game. A character will tell the user “I’m thinking of a number from 1 to 100. Try to guess it.” Each time the user makes a guess, he is told either “That’s it”, “Too high” or “Too low.” A smart user will make his first guess 50. (The midpoint between the low value and the high value, or average.) He starts with a low of 1 and a high of 100. If the guess is too low, he changes the high value to guess-1. If the guess is too high, he changes the high value to guess -1.

Note: *This game only works with integers because the number of possible values is finite.*

Example: The number is 64:

Low	High	Guess	Outcome	Action
1	100	50	“too low”	Change low to guess+1
51	100	75	“too high”	Change high to guess-1
51	74	62	“too low”	Change low to guess+1
63	74	68	“too high”	Change high to guess-1
63	67	65	“too high”	Change high to guess-1
63	64	63	“too low”	Change low to guess+1
64	64	64	“Yes!”	found=true, loop ends

```
public void myFirstMethod() {
    // A number guessing game
    // Programmer : Janet Joy
    Integer number = RandomUtilities.nextIntegerFromAToBInclusive( 1, 100 );
    this.bigBadWolf.think( "" + number, Think.duration( 1.0 ) );
    Integer guess = 0;
    Boolean found = false;
    Integer tries = 0;
    this.bigBadWolf.say( "I'm thinking of a number from 1 to 100." );
    this.bigBadWolf.say( "If you can guess it, I won't eat you. " );
    this.bigBadWolf.say( "You get 10 guesses." );
    while ( !found && tries <= 9 ) {
        tries = tries+1;
        guess = this.bigBadWolf.getIntegerFromUser( "Make a guess." );
        if( guess == number ) {
            // Setting found to true will end the loop
            found = true;
        } else {
            if( guess < number ) {
                this.bigBadWolf.say( "Too low." );
            } else {
                this.bigBadWolf.say( "Too high." );
            }
            this.bigBadWolf.say( "That was guess number " + tries );
        }
    }
}
```

```

    if( found ) {
        this.bigBadWolf.say( ( "You guessed it in " + tries ) + " tries." );
    } else {
        this.bigBadWolf.say( "Too bad, you couldn't guess it." );
    }
}

```

Let's add a very smart rabbit who adjusts high and low as described above.

```

public void myFirstMethod() {
    // A number guessing game
    // Programmer : Janet Joy
    Integer number = RandomUtilities.nextIntegerFromAToBInclusive( 1, 100 );
    this.bigBadWolf.think( "" + number, Think.duration( 1.0 ) );
    Boolean found = false;
    Integer tries = 0;
    Integer low = 1;
    Integer high = 100;
    this.bigBadWolf.say( "I'm thinking of a number from 1 to 100." );
    this.bigBadWolf.say( "If you can guess it, I won't eat you. " );
    this.bigBadWolf.say( "You get 10 guesses." );
    while ( !found && tries <= 9 ) {
        tries = tries+1;
        Integer total = low+high;
        Integer guess = total/2;
        this.bunny.say( "My guess is " + guess );
        if( guess == number ) {
            // Setting found to true will end the loop
            found = true;
        } else {
            if( guess < number ) {
                this.bigBadWolf.say( "Too low." );
                low = guess+1;
            } else {
                this.bigBadWolf.say( "Too high." );
                high = guess-1;
            }
            this.bigBadWolf.say( "That was guess number " + tries );
            this.bunny.think( ( ( "The number is between " + low ) + " and " ) + high
        );
    }
    if( found ) {
        this.bigBadWolf.say( ( "You guessed it in " + tries ) + " tries." );
    } else {
        this.bigBadWolf.say( "Too bad, you couldn't guess it." );
    }
}

```

Note: This is called a Binary search because each guess divides the number of possible values in half.