Frogger Lesson Plans
Table of Contents

January 2021

2345 Yale St., 1st floor
Palo Alto, CA 94306
info@codemonkey.com
www.playcodemonkey.com

TEACHER's GUIDE

LESSON Nº 1

Let's Get Started

Exercises 1-4

PART 1

Introduction

20 min.

In this Frogger game design course, your students will program a game by the same name, Frogger. Programming the game will introduce your students to basic programming and game-building concepts.

The objective of this game is to help the frog safely cross the road, without getting hit by a vehicle. The player controls the frog by swiping the screen. When a touch screen is not available, the arrows on the keyboard act as substitute to swiping the screen.

Objectives:

Within this lesson, students will:

- Write event handlers
- Write “rules” for different values of parameters (i.e. write if statements)
- Check parameters a function gets, and act according to their values
- Use a “forever loop”

Components:

- “onKey”, “step”, “if”, “setRotation”, “loop”

Part 1 - Introduction

Watch 5 min.

- Open this link in order to show your students the game that they will create at the end of the course.
- Ask your students to think of the different objects and characters they see in the game. Mention that these are called Sprites.
- Ask a volunteer to play the game.
- Ask for another two volunteers to play the game.

Part 1 - Introduction

Discussion 3 min.

Ask the following: What sprites were there in the game?
Make a list on the board.

The list should include the following:

- Frog
- Home
- Car
- Truck
- Heart
- Hedgehog

Part 1 - Introduction

Discussion 3 min.

Ask your students to think of the different properties that the sprites possess.

Make a list on the board.

The list should include the following:

- Position
- Direction
- Speed
- Size

Note:

Students may find some other properties too, such as color or proportion between width and length and so on.

The properties listed above, are properties that your students can manipulate in the game.

Part 1 - Introduction

Activity #1 3 min.

The following activity, facilitates the understanding of an event and an event handler.

1. Ask for two volunteers
2. Nominate one volunteer to be the "event causer"
3. Nominate the other volunteer to respond to an event (be the "event handler")
4. Decide with the "event causer" on at least three different events/sounds. For example: clapping, stomping, calling: "hello".
5. Agree with the "event handler" that he is "programmed" to respond to the clapping event only. His response to the clapping event should be raising his left hand.
6. Let the "event causer" cause events, while the "event handler" handles only the event she is programmed to handle.

Part 1 - Introduction

Activity #1 part 2 3 min.

This part of the activity, is a preparation for an if statement and parameter handling.

1. Add a "parameter" to the events. Each event will have a number of repetitions. Since we are interested in the clapping event, the parameter that will concern us will be the number of claps.
2. "Program" the event handler, to respond differently to the clapping event, according to the number of claps.

if one clap was heard,

raise your left hand

if anything other than one clap was heard (in other words: **else**) then

raise your right hand

Part 1 - Introduction

Activity #1 part 2 (con't) 3 min.

Ask the "events causer" to cause some more events, including one clap and two claps. Ask the "event handler" to handle the events agreed in accordance to the program she has.

Part 1 - Introduction

Activity #2 2 min.

The following activity facilitates the understanding of a sprite's rotation - specifically, that the rotation influences the direction of the sprite.

Ask for three volunteers.

Ask them to stand, each looking in different directions. Give them the following instructions:

1. move 4 of steps forward
2. turn 180 degrees
3. move 4 steps forward

Each student steps in the direction that they are facing, then turns around and steps the way back.

Part 1 - Introduction

Explain 1 min.

Each sprite has its own rotation. The rotation sets the direction in which the sprite moves.

When we want the sprite to move in a different direction, we need to change its rotation.

Part 1 - Introduction

Activity #3

5 min.

The following activity introduces the concept of loop, which is used when we need to repeat the **same set of instructions** over and over again.

Ask for an "instructor student" to give instructions (see next slide) to the other volunteers.

Write on the board to show how long and boring it is to repeat the same instructions over and over again.

- the need for a shorter way should arise
- while writing the last instructions, ask what would happen if you would have asked to step & turn three more times

Part 1 - Introduction

Activity #3

5 min.

1. move 4 of steps forward
2. turn 180 degrees
3. move 4 steps forward
4. move 4 of steps forward
5. turn 180 degrees
6. move 4 steps forward
7. move 4 of steps forward
8. turn 180 degrees
9. move 4 steps forward
10. move 4 of steps forward
11. turn 180 degrees
12. move 4 steps forward

Part 1 - Introduction

Activity #3

5 min.

Ask the instructor to give as precise and short instructions as possible to gain the same results as written on the board.

After completing the activity, ask the students to write these instructions using a loop:

```
Do the following 4 times:
    move 4 steps forward
    turn 180
    move 4 steps forward
```

Part 1 - Introduction

Explain 1 min.

A loop consists of 2 parts:

1. the loop's condition
    - which states the number of times the body of the loop should be repeated
2. the loop's body
    - contains instructions that will be repeated per the loop's condition

An important issue to remember about a loop is to take care and make sure the conditions of the loop is the desired one

- this is to avoid infinite loops

In today's lesson, your students will use a loop without condition. This means that the code inside the loop will run over and over for as long as the game runs.

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 10 min.

Ask your students to complete exercises 1 - 4.

Encourage students to read the instructions in each exercise, especially if they do not understand what needs to be done.

Use the classroom dashboard to keep track of student achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #2:

Ask your students to delete the **if** code from the @onSwipe event handler. Your code should be:

```
@onSwipe = (direction) =>

    @step 100
```

Run the game and swipe the screen in different directions. If you don't have a touch screen, use the arrows keys to simulate a touch screen. Notice that the frog moves up regardless of the swipe direction.

The argument of the direction is not checked, therefore the frog moves just by swiping the screen in any direction.

Now write the code of this exercise again:

```
@onSwipe = (direction) =>

    if direction == swipe.up

        @step 100
```

Run the game, swipe the screen in different directions. See that the frog moves up only when you swipe up (or press the up arrow key).

Part 2 – Let's Go!

Play Time 5 min.

Students should complete all tasks up to exercise 4.

Use this time to pass between your students and help them.

Encourage students who have completed all the exercises to solve the bonus task in exercise #4.

PART 3

Debriefing

5 min.

Part 3 – Debriefing

Discussion 5 min.

Discuss with your students what we have covered in this lesson:

1. What is an event handling functions?
    - A function that is called when the event happens and then does whatever you coded it to do
    - An example is the onSwipe function that is called when we swipe the screen
2. What happens if we do not write an event handling function?
    - The event will still occur, but nothing will happen
3. What are parameters for?
    - We can decide to do different things based on the parameter
    - For example - decide which way to rotate the sprite
4. What do we need to do to make the frog step to the right of the screen?
    - We need to change its rotation to 0 degrees
5. What do we need to do to in order to move the car up on the screen?
    - We need to change its rotation to 270 degrees

TEACHER's GUIDE

LESSON Nº 2

Shaping the Game

Exercises 5-8

PART 1

Introduction

25 min.

In this lesson your students will write an event handler for a new event: collision of two sprites.

Your students will update a sprite's property and add a new sprite.

Objectives:

Within this lesson, students will:

- Write an event handler for the collision event
- Change a sprite's property
- Set the sprite's position in the "world" of the game
- Add a new sprite

Components:

- "setX", "setY", "onCollide"
- Update "Collide world bound" property,
- Add new sprite

Part 1 - Introduction

Review 2 min.

Ask your students to recall the topics covered in the previous lesson.
Their answer should include:

- Event handling
- Acting according to the values of parameters passed to the event handler
- Forever loop

Discuss with your students: which event handling are you familiar with?
Their answers may include:

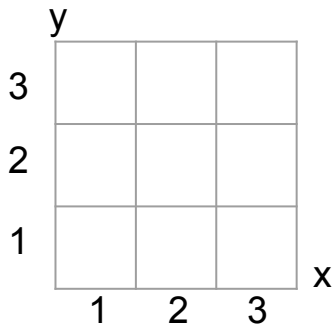
- Swipe event
- Keyboard event
- Mouse event
- Mouse click event

Part 1 - Introduction

Activity 5 min.

The following activity, facilitates the understanding of the x and y coordinates by playing Tic-Tac-Toe.

Draw a 3X3 grid:



Ask for three volunteers.

Two players and a drawer. The drawer will draw the moves on the grid. The players will give instructions to the drawer saying where to draw their moves. It is important that the instructions provided by the two players will be in the manner explained in the next slide.

Repeat the activity a few times to emphasize the coordinates.

Part 1 - Introduction

Activity (con't) 5 min.

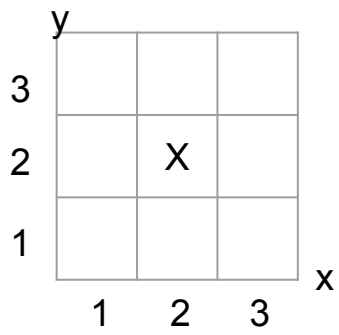
Instructions example:

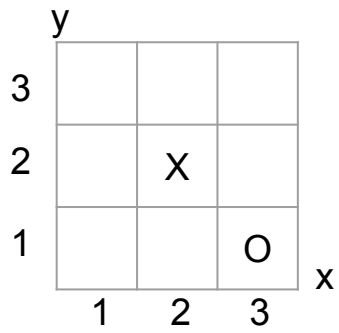
- player-X asks to place X at:
  - x = 2
  - y = 2

The drawer will draw X in the middle.

- player-O asks to place O at:
  - x = 3
  - y = 1

The drawer will draw O in the bottom right square.





Part 1 - Introduction

Explain 10 min.

The game's world has x and y coordinates. Placing a sprite into the game is done by setting its x and y coordinates.

Open Exercise #5:

Move the mouse around in the game world for a few seconds. Let your students notice that the coordinates, in the top left corner, change when the mouse is moved.
Move the mouse to a certain spot in the world and discuss the location of the mouse pointer in the world. Discuss the following topics:

1. where is 0,0?
    - top left corner
2. what is the world's width?
    - the highest value of x: 600
3. what is the world's height?
    - the highest value of y: 400

Part 1 - Introduction

Explain (con't) 10 min.

Move the mouse to the top left corner, where x and y are both 0.

Move the mouse to the bottom right corner to demonstrate the x and y values at that corner.

Notice that it is very difficult to get the mouse exactly to the corners. Therefore, the numbers will be close to (but not exactly) 0,0 in the upper left corner, or close to x=600, y=400 at the bottom right corner.

The world's size in the Frogger game is 0-600 horizontally and 0-400 vertically.

Part 1 - Introduction

Explain (con't) 10 min.

Click the **frog**'s settings icon and show the X and Y values.

Discuss how the X and Y values of the **frog** can be changed.

Change the X and Y values of the **frog** by writing a new value in the **frog**'s settings dialog box.

The values in the dialog box are used for the initializations of the game.

Therefore, the **frog**'s position in the world is changed the moment the values are changed in the dialog box.

Part 1 - Introduction

Explain (con't) 10 min.

Changing the sprite's position can also be done while the game is running by using:

```
@setX Xvalue  (for example: @setX 200)

@setY Yvalue  (for example: @setX 150)
```

These functions change the x and y positions of the sprite they refer to when the game gets to this code.

Referring to the sprite in which code is written in is done by the "@" sign. This will be explained more later in the lesson.

Part 1 - Introduction

Explain (con't) 10 min.

Chose the home sprite.

Write the following in the home sprite's code:

```
@setX 100
```

Run the game and show how the home "jumps" to its new X position.

Change the instruction to

```
@setY 100
```

Run the game and show how the home "jumps" to its new Y position.

Write both instructions, run the game and see the home "jumps" to its new position.

Show that when you stop the game, the home's position is set back to its initial position.

Part 1 - Introduction

Explain (con't) 10 min.

Elaborate on the need to address a specific sprite when writing an instruction that operates on a sprite.

Write the following in the Home sprite's code:

```
frog.setX 400

frog.setY 300
```

Run the game and show how the **frog** "jumps" to its new X and Y position.

Now, change the instructions to:

```
home.setX 400

home.setY 300
```

Run the game and show that the home "jumps" to its new X and Y position.

Part 1 - Introduction

Explain (con't) 10 min.

Change the last two instructions in the Home sprite's code to the following:

```
home.setX 400

@setY 300
```

Run the game and show how the **home** "jumps" to its new X,Y position.

Now change the instructions to:

```
@setX 400

@setY 300
```

Run the game and show that exactly the same results happen.

'@' is a way to refer to the sprite in which code is written in.

Part 1 - Introduction

Explain #2 8 min.

Elaborate on event handling furthermore:

Different events occur. Only events with event handlers are handled.

Open Exercise #6:

Run the game. Move the **frog** to collide with the car.

Discuss the following:

- What happens in our game if the car and **frog** collide?
  - Answer: nothing
- Why?
  - Answer: Because we did not write an event handler for that event.

The event where two sprites touch each other is a collision event.

Part 1 - Introduction

Explain (Con't) 8 min.

Checking for collision can be done by any of the two colliding sprites.

- the onCollide function can be defined in any of the two colliding sprites' code.

The definition of the onCollide function is specific for:

1. the sprite where the function is written
2. another (specified) sprite

The onCollide function is called when the two sprites collide, i.e. when the specific collision event occurs.

Part 1 - Introduction

Explain (Con't) 8 min.

The onCollide function gets the colliding sprites:

```
@onCollide sprite, () =>
```

The first argument is the "other" sprite.

The second argument is always the current sprite.
Empty parentheses () are the way to refer to the current sprite.

Part 1 - Introduction

Explain (Con't) 8 min.

Discuss the following:

What should be written and where should it be written in order to get the **frog** to move to x=320, y=360 when the **frog** and **car** collide?

Two answers are possible:

1. Defining the onCollide function in the car sprite
2. Defining the onCollide function in the frog sprite

Demonstrate the way to use both (explained in the following slide)

Part 1 - Introduction

Explain (Con't) 8 min.

1. chose the car sprite
2. in the car's code: write the code on the left side of the table below
3. run the game
4. cause the car and the frog to collide; stop the game
5. delete the code from the car's sprite code
6. chose the frog sprite
7. in the frog's code: write the code on the right of the table below
8. run the game
9. cause the car and the frog to collide

in the **car**'s code	in the **frog**'s code

`@onCollide frog, () =>`
`  frog.setX 320`
`  frog.setY 360`
	`@onCollide car, () =>`
	`  @setX 320`
	`  @setY 360`

PART 2

Let’s Go!

15 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 5 min.

Ask your students to complete exercises 5 - 8.

Encourage students to read the instructions in each exercise.

Use the teacher dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #6:

One of the properties of a sprite is: *Collide world bound*.

It defines whether the sprite can step beyond the boundaries of the world or not.

Run the game and move the **frog** to one of the boundaries of the world.
Emphasize that when the **frog** collides with the boundary of the game, it becomes stuck and steps in place.

Chose the frog's sprite and open its settings.
Uncheck the property *Collide world bound*.

Run the game and show your students how the **frog** continues to step and disappear.

Part 2 – Let's Go!

Play Time 5 min.

Students should complete the tasks up to exercise 8.

Use the time to pass between your students and help them.

Encourage students who complete all exercises to solve also the bonus task in exercise #8.

PART 3

Debriefing

5 min.

Part 3 – Debriefing

Discussion 5 min.

Discussion: what was covered in this lesson

1. What will happen, if the following code appears in the home's code?

```
@onCollide frog, () =>
    @setX 320
    @setY 360
```

   - When the **frog** and home collide, the home's position will be changed to x = 320 and y = 360

1. What should be written in the **home**'s code, to position the **frog** on the left side of the screen?
   - `frog.setX 0`
2. Which coordinates can be given to the car and where should they be written in order for the car to not be seen on the screen when the game starts?
   - Set the value of the car's X property, in the dialog box, to be less than zero

Demonstrate these ideas.

TEACHER's GUIDE

LESSON Nº 3

Properties for All

Exercises 9-11

PART 1

Introduction

15 min.

In this lesson, your students will utilize the different sprite properties that they previously learned about. They will also learn different ways to change the specific properties of sprites.

For example properties can be changed the following ways:

- Either only from the sprite's settings
- Either only from the code
- Both from the sprite's settings and code.

Objectives:

Within this lesson, students will:

- Change sprite's properties
- Set the speed and scale of a sprite randomly

Components:

- "setScale", "setSpeed", "random"

Part 1 - Introduction

Review 3 min.

In the previous lesson, your students changed a few properties of sprites.

Ask your students to list these properties:

1. Collide world bound
   a. Defines whether a sprite can “step” beyond the world bound or not
2. Position of a sprite, the position is combined of two properties:
   a. X - the x position of a sprite
   b. Y - the y position of the sprite

Part 1 - Introduction

Review(cont') 3 min.

Two ways to change the sprite's properties were used.

The sprite's position was changed by:

1. the sprite's settings
2. using the two functions:
    - setX
    - setY

In today's lesson, your students will continue to explore the properties of sprites.

Part 1 - Introduction

Explain 12 min.

Sprites have different properties. These properties define the way the sprite looks and moves in a game.

Some properties can be changed from the sprite's setting icon. These changes are updated immediately.

For example:

- Collide world bounds

Some properties we can be changed during the game based on things that happen during the game.

In order for us to change properties during the game, we need to add code that changes the properties. We will use various functions to change these properties.

Part 1 - Introduction

Explain (con't) 12 min.

For example:

- setSpeed - to change the sprite's speed
- setScale - to change the sprite's scale

Some properties can be changed both from the sprite's settings and from the code.

For example:

- Position of the sprite
  - X,Y - from the settings icon
  - setX, setY - from the code
- Rotation of the sprite
  - Rotation - from the settings icon
  - setRotation - from the code

Part 1 - Introduction

Explain (con't) 12 min.

Open Exercise #8:

Click on different sprites shown in this exercise. Show your students how each sprite has its own set of properties that can be found in either the code itself or the icon settings, or both in some cases. Emphasize that each sprite has properties with different values.

For example, you can point out that:

In the frog's sprite:

- Collide world bounds is checked - the sprite cannot step beyond the world
- The initial rotation is 270 - we change it during the game using the setRotation function (depending on the direction of the swipe)

Part 1 - Introduction

Explain (con't) 12 min.

In the car's sprite:

- Collide world bounds is unchecked
- The initial rotation is 0 and is not changed during the game
- Show is checked - the sprite is shown
  - You can uncheck it - the car is not shown in the game but it is still there even if we don't see it
  - Ask your students what will happen when we run the game?
    - the car will not **step**
    - the collide event will happen
  - You can run the game and move the frog to the following:
    - where the car is - the frog is moved to the beginning (onCollide function is called)
    - where the car should have moved to - nothing happens because the car did not "step"

Part 1 - Introduction

Explain (con't) 12 min.

In the home's sprite:

- Immovable is checked - this means that when another sprite collides with this sprite, the sprite (the home in this case) will not move
  - You can run the code and show the difference when the frog collides with the car (the car is pushed up) vs. what happens when the frog collides with the home (the home does not move)

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 10 min.

Ask your students to complete exercises 9 - 11.

Encourage your students to also read the instructions in each exercise, especially if they do not understand what needs to be done.

Use the teacher dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #10:

In this exercise we use the random function.

This function gets two arguments:

- minimum value
- maximum value

The function returns a random number between the minimum and the maximum.

The syntax of the function is:

```
random minimum, maximum
```

For example, if we write the following code

```
random_var = random 0, 5
```

Then each run random_var will be a different number between 0 and 5.

Part 2 – Let's Go!

Play Time 5 min.

Students should complete up to exercise 11.

Use the time to pass between your students and help them.

Encourage students who complete all exercises to solve also the bonus task in exercise #11.

PART 3

Debriefing

10 min.

Part 3 – Debriefing

Discussion 10 min.

Discuss with your students what we covered in this lesson:

Ask your students to:

1. Make the frog half its size when it collides with the car sprite
   - instead of setting its position to the beginning
2. Make the frog regular size when it collides with the home sprite

car's code (both cars):

```
@onCollide frog, () =>
    frog.setScale 0.5
loop
    @step 800
    @setX -100
```

home's code:

```
@onCollide frog, () =>
    frog.setScale 1
```

Part 3 – Debriefing

Discussion 10 min.

Note the following regarding “setScale” and “setSpeed” functions:

1. These functions apply the changes per the original scale and speed of the sprite
   - For example:

```
frog.setScale 2 # frog will be twice as big

frog.setScale 2 # nothing will happen
```

   - Or if we code:

```
frog.setScale 2 # frog will twice as big

frog.setScale 3 # frog will be three times as big (from its original scale)
```

2. The functions can get an argument less than 1 and then the scale is smaller and the speed is slower
3. When the argument is 1, the sprite is changed to its original scale and speed

TEACHER's GUIDE

LESSON Nº 4

Beyond Sprites

Exercises 12-16

PART 1

Introduction

15 min.

This lesson will introduce various game effects that can make the game more challenging and fun.

Your students will explore the different tabs other than the Sprites tab.

Your students will also be introduced to a new entity called the Counter widget.

Objectives:

Within this lesson, students will:

- Change the size of the world
  - set which is the main sprite to follow
- Duplicate sprites
- Add sounds to the game
- Use a Counter widget

Components:

- Duplicate sprite, Counter widget, Sounds

Part 1 - Introduction

Review 3 min.

Up until now we explored the sprites.

Ask your students what do we know about sprites? What can we do with them?

- Add Sprites
- Move the Sprites
- Handle Events
  - Do something when we swipe the screen
  - Do something when two sprites collides
- Change properties
  - position
  - rotation
  - scale
  - speed

Part 1 - Introduction

Explain 12 min.

In today's lesson we will learn about the other three tabs in the game.

We will start with the **Game** tab.

In this tab, properties of the game are defined.

The main properties are:

- World's size - the default size of the world is what we see on the screen. This is defined by two properties:
  - World's width - default is set to 600
  - World's height - default is set to 400
- Camera target - when the size of the world is bigger than the default, we want to follow one of the sprites so when it moves, the "camera" follows it.
- Gravity - when this is zero (as in this course) it means that there is no gravity in the world. The Platformer course has Gravity.

Part 1 - Introduction

Explain (con't) 12 min.

Open Exercise #11:

Go to the Game tab and show your students the various properties there.

Change the following:

- World width to 1200
- World height to 800

Run the game and move the frog to the right and down. It will disappear and we will not see where it moves.

Now, stop the game and choose to follow the frog:

- In the Camera target choose frog (instead of None)

Run the game again and see how when the frog moves to the right and down we can follow it.

Part 1 - Introduction

Explain (con't) 12 min.

The **Sounds** tab includes sounds that we can add to the game.

When we add a sound, we need to name it.

The sound has one function called play, which plays the sound.

Go to the Sounds tab.

This tab shows the sounds we have in the game. When we press on the + green button we see the different sounds that we can add to the game. Show your students that we can play the sounds before we add them in order to chose the one we want.

Ask your students when we can add sounds?

For example:

- When the frog collides with a car
- When the frog collides with the home

Part 1 - Introduction

Explain (con't) 12 min.

The last tab is the **Widgets** tab.

Widgets are objects we can add to the game. Each widget has a specific functionality. It does not "step" like the sprites.

Examples of Widgets: Counter, Text, Timer, Clock, Dialog, Button.

The Widgets do not have many properties. All of them have the Show property - we can decide if we want to show them or not.

Each widget also has its own, unique properties.

We will learn about most of these widgets later on in the course. For now, we will focus on the Counter widget.

Part 1 - Introduction

Explain (con't) 12 min.

The Counter widget is used mainly to count something in the game.

Ask your students if they can think of examples of something we want to count?

- score
- lives
- collected items

The counter widget has a property value which stores the value of the counter.

The default is set to 0, but can also be changed.

If we have a counter named count, then we can refer to it as:

```
count.value
```

Part 1 - Introduction

Explain (con't) 12 min.

Usually, we will want to increase the value of the counter. Sometimes we will want to increase it by 1, for example, if we count the items we collected. Other times we will want to increase it by 100, for example, if we want to add 100 points to the score each time the frog gets home.

We use the operator **+=** to increase a value.

This operator adds the number on the right to the variable on the left.

For example:

```
var += 5
```

This adds 5 to the variable **var**. If its original value was 0, then now it is 5. If its original value was 10, then now it is 15.

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 10 min.

Ask your students to complete exercises 12 - 16.

Encourage your students to also read the instructions in each exercise, especially if they don't understand what needs to be done.

Use the classroom dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #13:

In this exercise we will duplicate sprites. This functionality is very useful when we want to have many occurrences of the same sprite with the same functionality.

Ask your students to give you examples of a sprite we want to duplicate:

- cars and trucks
- bananas for the monkey to collect

In our game, each time the frog and any car or truck collides, we want the frog to be set back to its beginning position.

If we write the onCollide function in the frog sprite, each time we duplicate the sprite we will still need to add another onCollide function (i.e. car1, car2).

When we write the onCollide function in the car sprite, each time we duplicate the sprite, the code for the onCollide with the frog will already exist and we will not need to add or change the code.

To duplicate a sprite, open the sprite's settings and click on the duplicate icon 

Part 2 – Let's Go!

Play Time 5 min.

Students should complete up to exercise 16.

Use the time to pass between your students and help them.

Encourage students who complete all exercises to solve also the bonus task in exercise #14 and in exercise #16.

PART 3

Debriefing

10 min.

Part 3 – Debriefing

Discussion 10 min.

Let us use what we learned in today's lesson to create a mini game.

Tell your students you want to create a race between the cars and trucks. Which vehicle will loop around the most times before the frog gets home? The car or the truck?

What do we need to do?

1. Add a counter to count the cars - countCars
2. Add a counter to count the trucks - countTrucks
3. Update the counter inside the loop (after setX)
    a. in the car sprite - update countCars
    b. in the truck sprite - update countTrucks
4. Duplicate the car sprite and truck sprite at least once

You can look at the following link for an example.

Part 3 – Debriefing

Discussion 10 min.

You can use one of the exercises solved in this lesson - for example Exercise #13.

You can also click on Create Games and build a new game for this purpose.

You need to make the following changes:

Game tab:

- World Height is 800
- Camera target is frog
- Gravity is 0

Widgets tab:

- Add counter and name it countCars
  - place it on the bottom left side
- Add counter and name it countTrucks
  - place it on the bottom right side

Part 3 – Debriefing

Discussion 10 min.

Sprites tab:

- Add a car sprite and name it car1
- Write the following code in car1:

```
@onCollide frog, () =>
      frog.setX 320
      frog.setY 750
loop
    @setScale random 0.5, 1.5
    @setSpeed random 0.7, 1.5
    @step 800
    @setX -100
    countCars.value += 1
```

Part 3 – Debriefing

Discussion 10 min.

Sprites tab:

- Add a truck sprite and name it truck1
- Write the following code in truck:

```
@onCollide frog, () =>
      frog.setX 320
      frog.setY 750
loop
    @setScale random 0.5, 1.2
    @setSpeed random 0.7, 1.5
    @step 800
    @setX -100
    countTruck.value += 1
```

Part 3 – Debriefing

Discussion 10 min.

Sprites tab:

- Duplicate car1 twice
  - so you have three cars all together
- Duplicate truck1 twice
    - so you have three trucks all together
- Place the cars and trucks along the road
- The code in the frog sprite is the same as the previous exercises

Now start playing and see the counters increase as the cars and trucks pass the right borders.

TEACHER's GUIDE

LESSON Nº 5

This is Better

Exercises 17-21

PART 1

Introduction

20 min.

This lesson defines the functionality for losing in the game.

We will use a function to handle what happens when the frog gets hit by a car or a truck.

Your students will use another Counter to count the lives of the frog.

Depending on the value of the counter, we will return the frog to the beginning or destroy it and thus end the game.

Objectives:

Within this lesson, students will:

- Define a function and call it from different sprites
- Use a Text widget
- Add functionality of lives and losing a live

Components:

- “function”, “destroy”, “if else”
- Text widget

Part 1 - Introduction

Review 2 min.

In the previous lesson we discussed other objects in the game besides sprites.

We reviewed the various tabs:

- Widgets
  - Object with specific functionality
  - Examples: counter, text, clock, timer, dialog
- Sounds
  - Add sounds to the game
  - Use the function play() to play the sounds
- Game
  - Define the world's size
  - Define which sprite the camera follows

Part 1 - Introduction

Activity 8 min.

The following activity facilitates the understanding of using functions.

Ask for eight volunteers. Divide them into two groups.

In each group choose a student who is the “programmer”.

The other three volunteers will be the “sprites” that execute the code.

The programer will write the code for the volunteers on the whiteboard.

Tell the “programmer” that the functionality of the sprite is to walk 5 steps, turn around and step 5 more steps.

They can use the following pseudo-code:

```
walk 5

turn around

walk 5
```

Part 1 - Introduction

Activity (con't) 8 min.

The difference between the two groups will be the following:

- Group 1
  - the code is written in each sprite.
- Group 2
  - use a function with the code of this functionality and call the function from each sprite

Before starting, ask your students which option they think is the prefered way.

They should answer that group 2 is the better way to code.

Divide the whiteboard into 2 so each "programmer" will have their own space to write the code.

Part 1 - Introduction

Activity (con't) 8 min.

Group 1

Sprite 1-A	Sprite 1-B	Sprite 1-C
walk 5	*walk 5*	*walk 5*
turn around	*turn around*	*turn around*
walk 5	*walk 5*	*walk 5*

Part 1 - Introduction

Activity (con't) 8 min.

Group 2

```
function to_do:
    walk 5
    turn around
    walk 5
```

Sprite 2-A	Sprite 2-B	Sprite 2-C
`to_do()`	`to_do()`	`to_do()`

Now, ask all the volunteer to execute their codes.

All of them should do exactly the same.

Part 1 - Introduction

Activity (con't) 8 min.

Now, we want each sprite to jump before they start walking and after they return.

Ask the "programmers" to add this functionality.

After the change, their code should like the following:

Group 1

Sprite 1-A	Sprite 1-B	Sprite 1-C
jump once	*jump once*	*jump once*
walk 5	*walk 5*	*walk 5*
turn around	*turn around*	*turn around*
walk 5	*walk 5*	*walk 5*
jump once	*jump once*	*jump once*

Part 1 - Introduction

Activity (con't) 8 min.

Group 2

```
function to_do:
    jump once
    walk 5
    turn around
    walk 5
    jump once
```

Sprite 2-A	Sprite 2-B	Sprite 2-C
`to_do()`	`to_do()`	`to_do()`

Part 1 - Introduction

Explain 10 min.

Today we will learn about functions.

A function is a set of instructions that performs a specific task.

Ask your students to give you examples of functions:

- step
- setRotation
- setX
- setY
- setSpeed
- setScale

The computer will execute the function only if we call it from our code.

Part 1 - Introduction

Explain (con't) 10 min.

A function can get one or more arguments, or no argument at all.

Functions with arguments:

- setX
- setY
- setSpeed
- SetScale
- random (gets two arguments - minimum and maximum)

Functions without arguments:

- play (for sounds)
- destroy

Part 1 - Introduction

Explain (con't) 10 min.

A function can return a value or not.

Example of functions that do not return a value:

- setX
- setY
- setSpeed
- SetScale
- play

Example of functions that return a value:

- random
- getX (returns the x position of a sprite)
- getY (returns the y position of a sprite)

Part 1 - Introduction

Explain (con't) 10 min.

The syntax for defining a function:

```
functionName = (argument) =>
```

Example of defining a function with an argument:

```
whereTo = (x) =>
    @setRotation x
    @step 100
```

Example of defining a function that returns a value:

```
myCalc = (x, y) =>
     return x + y
```

Part 1 - Introduction

Explain (con't) 10 min.

Now that we know what functions are and how to define them, we need to understand why we should use them.

Ask your students to list reasons why functions are beneficial. Their answers should include:

- Once we write the function, we can call it from any place in the code
  - we do not need to repeat code
- If we want to add more code, we only need to add it once
- When we call a function, we do not need to know how it works
  - just its name, arguments and return value
- It is easier to understand the code once it is broken down into functions

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that were covered in previous exercises.

Part 2 – Let's Go!

Play Time 10 min.

Ask your students to complete exercises 17 - 21.

Encourage your students to also read the instructions in each exercise, especially if they don't understand what needs to be done.

Use the teacher dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #20:

In this exercise, we will use a new function called destroy.

This function deletes the sprites from the game.

We have a counter that counts the lives of the frog. When the value of the counter is 0, it means that the frog lost all of its lives and we delete (destroy) it.

Once the frog is deleted, our game ends.

The syntax to use this function is:

```
@destroy()  # When we want to destroy the sprite in which the code is written in
```

Part 2 – Let's Go!

Play Time 5 min.

Students should complete up to exercise 21.

Use this time to pass between your students and help them.

Encourage students who complete all exercises to solve also the bonus task in exercise #18 and #21.

PART 3

Debriefing

5 min.

Part 3 – Debriefing

Discussion 5 min.

Let us add more homes. We want the frog to only get to each home once!

Ask your students to:

1. Define a new function in the frog's code
    - Name it atHome
    - Call it when the frog collides with the home
        i. The function should have the same code as the onCollide function in the home sprite
2. Destroy the home after the frog collides with it
    - We can not add this in the function because the function is general and to destroy a sprite we need to specify the sprite name
3. Duplicate the home three times and place the home sprites at the top of the screen

Part 3 – Debriefing

Discussion 5 min.

The function in the frog's code:

```
@atHome = () =>

    @setX 320

    @setY 360

    homeSound.play()

    score.value += 100
```

The code of the home's code:

```
@onCollide frog, () =>

    frog.atHome()

    @destroy()
```

You can add this to Exercise #21.

This is an example of the how game should look like.

TEACHER's GUIDE

LESSON Nº 6

It is Time to Move on

Exercises 22-26

PART 1

Introduction

15 min.

This lesson will introduce another widget called Timer.

We will add another obstacle for the frog to get home - a hedgehog.

The hedgehog is at “home” and then moves away from it. We will use the timer widget to move it back and forth.

Objectives:

Within this lesson, students will:

- Use a heart sprite to add lives to the frog
- Recap what we previously learnt
- Use a Timer widget
- Add functionality for hedgehog

Components:

- “getX”
- Timer widget

Part 1 - Introduction

Review 2 min.

In the previous lesson we discussed functions.

Functions can:

- have none, one or more arguments
- return a value or not

We defined a new function and called it **hit**. The function handles what happens when the frog is hit by a car or truck. We called the function from each car and truck sprites. We will use it also in today's lesson.

The syntax to define a new function is:

```
functionName = (argument) =>
```

Part 1 - Introduction

Explain 13 min.

So far, the only obstacle that the frog has is the car or truck sprites.

We will now add another obstacle - the hedgehog.

The hedgehog can be either:

- At home
  - its x position is the same as the home's x position
- Disappear
  - its x position is -50 which is out of the game

The hedgehog moves back and forth between these two positions.

If the frog gets home when the hedgehog is there, then the frog loses a life and is set back to the beginning or destroyed (depending on how many lives it still has).

Part 1 - Introduction

Explain (con't) 13 min.

To check if the hedgehog is home, we will use the == operator.

We need to know the hedgehog's position.

We will use the function getX. This function returns the x position of the sprite.

We have a hedgehog sprite that we want to move to the home's position (in our game x = 400) and then move it away from the game (in our game x = -50).

Ask your students the following: When we want to make the hedgehog disappear, should we use the destroy function?

The answer is no. The reason is that once we destroy the sprite, we will no longer be able to move it back home.

Part 1 - Introduction

Explain (con't) 13 min.

So how can we move the hedgehog back and forth?

We will do it using the Timer widget.

When we add a timer widget, we need to start it and specify how many seconds we want it to run for.

For example, if we want it to run for 5 seconds, we will code:

```
timer.start 5
```

We can start the timer in the timer's code or from any other sprite or widget.

When we start it from the timer's code, the timer will start running the seconds once we run the game.

When we start the timer from any other sprite/widget code, the timer will start running when we get to that code.

Part 1 - Introduction

Explain (con't) 13 min.

Note, that if we start the timer in another sprite/widget not inside a function, the timer will also start running when we run the game.

Here are a few examples:

Open Exercise #21

First, add a Timer widget.

Go to the Widgets tab and choose timer, we will name it timer (the default name).

In the timer's code write:

```
@start 3
```

Run the game and show your students that the timer start running and counts three seconds backwards until zero.

Part 1 - Introduction

Explain (con't) 13 min.

Delete starting the timer from the timer's code and go to the frog's code. Start the timer there. Remember that here we need to write timer.start since we are writing the code in frog's sprite (and not in the timer's code).

```
timer.start 3

@hit = () =>

      carHit.play()

      …
```

Run the game. The timer will start when we run the game.

Now move the start of the timer to be inside the hit the function.

```
@hit = () =>

      timer.start 3

      carHit.play()

      …
```

Part 1 - Introduction

Explain (con't) 13 min.

Before running the game, ask your students what they think will happen now?

The answer is that the timer will start when the frog gets hit by a car or truck. It will not restart if the frog gets hit before the timer ends.

The timer can also run in a loop, which means that once it ends it automatically starts again. The start function can get two arguments:

- number of seconds
- loop mode
  - no - the default (we do not need to specify)
  - yes - will run the timer in a loop

Go back to our example and change the timer to run in a loop:

```
timer.start 3, yes
```

Part 1 - Introduction

Explain (con't) 13 min.

Run the game, get the frog hit by a car (once) and see that the timer starts over once it ends.

The Timer widget has another function. This is an event function which is called each time the timer ends. The function is called onEnd and it is already defined when we add a Timer widget.

Go to the Widget tab and click on the timer widget that we added. Show your students that the function onEnd is defined there.

```
@onEnd = () =>

    # Your code here
```

If we do not write any code inside this function then nothing will happen when the timer ends.

If the timer runs in a loop, the onEnd function is called each time the timer ends.

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 5 min.

Ask your students to complete exercises 22 - 26.

Encourage your students to also read the instructions in each exercise, especially if they do not understand what needs to be done.

Use the teacher dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

Open Exercise #23:

This is a special exercise that recaps everything we have learned until now.

Briefly discuss each topic we have covered so far in the course:

- move the frog in the direction of the swipe
  - defining the function onSwipe
  - using the function setRotation to change the frog's rotation
- add a sprite and duplicate it
  - we added car and truck sprites
  - we duplicated a sprite from the sprite's settings
- set the size and speed of the cars randomly
  - setSpeed
  - setScale
  - random
- used the camera to follow the frog
  - from the Game tab, choosing frog in the Camera target dropdown

Part 2 – Let's Go!

Walkthrough 5 min.

- add sound to our game
  - from the Sounds tab
  - use the function play to play the sound
- score points when the frog got home
  - use the Counter widget and named it score
  - the value of the counter is stored in the property value (score.value)
  - use the operator += to add points
  - points are added when the frog gets home
- count the lives of the frog
  - use widget counter and name it lives
  - use the operator -= to reduce life
- define a function (hit) to handle what happens to the frog when it is hit by a car/truck sprite
  - when lives count is 0, destroy the frog
  - if there are more lives (lives is not 0) then set the frog to the beginning

Part 2 – Let's Go!

Play Time 10 min.

Students should complete up to exercise 26.

Use the time to pass between your students and help them.

Encourage students who complete all exercises to solve also the bonus task in exercise #24 and #26.

PART 3

Debriefing

10 min.

Part 3 – Debriefing

Discussion 10 min.

It is a race against time.

Let us add a timer and if the frog does not get home by the time the timer ends, we will move the frog to the beginning.

Count how many times the frog gets home before the timer ends and if it is more than twice, we will add another life to the frog.

You will need to add two widgets:

1. Timer widget - let us name it homeTimer
2. Counter widget - getHome

Part 3 – Debriefing

Discussion 10 min.

homeTimer's code:

```
@start 15, yes
@onEnd = () =>
    if getHome.value == 0  # If we didn't get home we move the frog back to the beginning
        frog.setX 320
        frog.setY 360
    if getHome.value > 1  # If the frog got home twice or more, then add a life
        livesCount.value += 1
    getHome.value = 0  # Each time the timer ends, we set the counter to 0 to start counting again
```

Part 3 – Debriefing

Discussion 10 min.

home's code:

```
@onCollide frog, () =>
      frog.setX 320
      frog.setY 360
      score.value += 100
      homeSound.play()
      getHome.value += 1   # This is the line we need to add
```

You can add this to Exercise #26.

This is an example of how the game should look like.

TEACHER's GUIDE

LESSON Nº 7

Game Over

Exercises 27-30

PART 1

Introduction

15 min.

This lesson introduces two widgets - the Clock and Dialog (referring to interface pop-up messages).

We will add bonus points if the frog gets home in less than eight seconds.

We will display a message to the user with the score when the game ends.

Objectives:

Within this lesson, students will:

- Use A Clock widget
- Add bonus points
- Use a Dialog widget
- Reset game

Components:

- “reset”
- Clock widget, Dialog widget

Part 1 - Introduction

Review 2 min.

In the previous lesson, we introduced two new sprites:

1. heart - to add a life to the frog
2. hedgehog - another obstacle for the frog

We also used a new widget:

1. Timer

We used the Timer widget to move the hedgehog back and forth. We started the game when the hedgehog was “hidden”.

We started the timer repeatedly. Each time the timer ended, the hedgehog was either moved home or “hidden”, depending on its current position.

Part 1 - Introduction

Review (con't) 2 min.

Ask your students what the timer's functions are:

- start
  - start the timer for x seconds

    ```
    start 5
    ```

  - specify whether we want the timer to run in a loop

    ```
    start 5, yes
    ```

- onEnd
  - function is called when the time ends
  - function is already defined in the Timer widget, user can add code or not

    ```
    @onEnd = () =>
    ```

Part 1 - Introduction

Explain 13 min.

Today we will use another two widgets:

1. Clock
2. Dialog

Let us start with the Clock widget.

The Clock widget is similar to the Timer widget. The difference is that the Clock widget starts from 0 and counts how many seconds have passed whereas the Timer widget counts seconds backwards until it gets to 0.

The Clock widget has a function start, this function does not need arguments.

```
clock.start()
```

Part 1 - Introduction

Explain (con't) 13 min.

Open Exercise #26

Go to the Widgets tab and add a Clock widget.

Start the clock. This is the default code for the Clock widget, so we actually do not need to add anything here.

Clock's code:

```
@start()
```

Run the code and show your students that the clock starts running.

Ask the your students what will happen if we also start the clock in the hit function?

The answer is that each time the hit function is called (the frog gets hit by a car, a truck or the hedgehog) the clock will start counting from 0.

Part 1 - Introduction

Explain (con't) 13 min.

Frog's code:

```
@hit = () =>
      clock.start()
      livesCount.value -= 1
      …
```

Run the code and show how each time the frog gets hit, the clock starts counting from 0 again.

The clock has another function getSeconds. This function, which does not get an argument, returns the number of seconds that passed since the clock started running.

We can use the operator <, > or == in an if statement to check how many seconds have passed and do something accordingly.

Part 1 - Introduction

Explain (con't) 13 min.

Ask your students to give you examples:

- if the frog gets home in less than x seconds we give a bonus
  - add y points to the score
- if the frog gets home in more than x seconds we give a penalty
  - reduce z points from score

In our game, we give bonus points if the frog gets home in less than eight seconds.

Now, we will learn about the Dialog widget.

We use the Dialog widget to display messages to the user.

We can use it for example:

- Give instructions before we start the game
- At the end of the game

Part 1 - Introduction

Explain (con't) 13 min.

The Dialog widget has a property called text, which holds the text that is displayed to the user.

The Dialog widget is used to display messages during the game. For example, before the game is started or when it ends.

The text can be updated from two places:

1. the widget's settings
2. the code of any sprite/widget

The Dialog widget has an event function onConfirm, which is called once the button is clicked.

PART 2

Let’s Go!

20 min.

Course Structure:
The Course is divided into exercises. Each exercise includes tasks that need to be completed. Upon successful completion of an exercise, the next exercise becomes available.

Each task ends with clicking the “check” button, after making the changes in the code, as required in the task. If the code is correct, the next task becomes available, if not, the task is marked as “failed” and retry is needed.

When Check button is not available, the code should be tested by using the Run button.. Usually when running the code, The player is asked to help the frog get home.

Bonus Tasks

Encourage your students to complete these as well. Bonus tasks are more complicated and most of them are not step by step instructions like . in the tasks within the exercises.

Some bonus tasks include topics that ere covered in previous exercises.

Part 2 – Let's Go!

Play Time 10 min.

Ask your students to complete exercises 27 - 30.

Encourage your students to also read the instructions in each exercise, especially if they don't understand what needs to be done.

Use the classroom dashboard to keep track of students' achievements.

Part 2 – Let's Go!

Walkthrough 5 min.

[Open Exercise #29]

In this exercise, we want to show the user how many points they scored in the game.

We need to add the variable that holds this information to the text of the dialog - in our game it is score.value.

The syntax to add a variable to the text is:

```
dialog.text = "You scored: #{score.value} points!"
```

When the code runs, the computer replaces the variable with its value and if for example score.value is 300, then we will get a message:

You scored: 300 points!

Part 2 – Let's Go!

Play Time 5 min.

Students should complete up to exercise 30.

Use this time to pass between your students and help them.

Encourage students who completed all exercises to also solve all bonus tasks in the course.

PART 3

Debriefing

10 min.

Part 3 – Debriefing

Discussion 10 min.

Ask your students how we can add a life if the frog got home in less than five seconds without getting hit by a car or truck?

In the home code we will need to check if the seconds on the clock are less than five, if yes then add 1 to livesCount. Of course, we also need to start the clock again.

In the frog's code we need to update the hit function to also start the clock again. This is because we want the frog to get home without getting hit by a car or a truck.

Ask your students why we can not use a timer and start it for five seconds instead?

The answer:

1. We can not stop a timer and start it again
2. When the frog gets home we do not know how many seconds have passed

You can use Exercise #27 or look at the following link as an example.

Part 3 – Debriefing

Discussion 10 min.

Add a Clock widget, name it clock and start it.

clock's code:

```
@start()
```

home's code (changes are in bold):

```
@onCollide frog, () =>
      frog.setX 320
      frog.setY 360
      score.value += 100
      homeSound.play()
      if clock.getSeconds() <= 5
          livesCount.value += 1
      clock.start()
```

Part 3 – Debriefing

Discussion 10 min.

hit function in the frog's code (changes are in bold):

```
@hit = () =>
      livesCount.value -= 1
      bump.play()
      if livesCount.value == 0
            @destroy()
      else
            @setX 320
            @setY 360
      clock.start()
```

Part 3 – Debriefing

Discussion 10 min.

Ask your students how by using only one dialog we can display the user two different messages:

1. start the game
2. game over

The answer is that we can update the property text in any sprite or widget in the game.

You can use Exercise #27 or look at the following link as an example.

Add a Dialog widget and name it dialog.

dialog's code:

```
@text = "Get the frog home safely!"

@onConfirm = () =>

    @hide()
```

Part 3 – Debriefing

Discussion 10 min.

hit function in the frog's code (changes are in bold):

```
@hit = () =>
      livesCount.value -= 1
      bump.play()
      if livesCount.value == 0
            @destroy()
            dialog.text = "Game Over! You scored: #{score.value} points."
            dialog.show()
      else
            @setX 320
            @setY 360
```

GREAT JOB!