Case Study #5 - Exercises

Exercise #1

Assumptions

1. We are going to use the code from our previous exercise: *"Exercise Solutions\component_004[Collision_Detection]\component_004_002"* and we are simply going to add the following logic:
    a. We are going to create a **`Timer`** which will count down from a 100 (we will use an interval timer utility class here with an interval of 1 second.)
        i. At each "tick" we will create a random position, random direction (which will generate a direction in the range of [-1, 1], as well as random speed (both based on the utility class that we created in *"Exercise Solutions\component_001[Square Shape]\component_001_001"*
        ii. We will then create a new ball with the above parameters.
    b. We will add a new variable to the MyCollidable class which will represent the current life: **`objectLifeValue.`**
    c. Then, at each collision (with other ball or the wall) we will subtract 1 from the life value and add some code that will check:
        i. If the **`objectLifeValue`** is 0 then we remove the ball from the game.

Solution:

1. We are basically re-using the code from previous exercises and we just add a few tweaks into existing code.
2. We will use a **`Timer`** utility class, so we will need to add code to **`update()`** method of the main game class.
3. We are going to test the code with a timer that will be an interval timer set to a 10, as this will be a more time-efficient way to test our code.

You will find the code solution in the directory *"Exercise Solutions\component_005[Timers]\component_005_001"*

Exercise #2

Assumptions:

1. We are going to add, for each ball, a **`TextComponent`** which will display the current life number in red. We will simply composite the **`TextComponent`** with the **`MyCollidable`** object.
    a. As you remember only components can be added as children of the Flame Engine component tree hierarchy, so we will need to build this TextComponent ourselves with our own formatting.
    b. We will create a separate class by extending TextComponent and adding the necessary data:
        i. Ordinal number of the ball.
        ii. Health of the ball
2. In this exercise we will use the **`TimerComponent`** class instead of the utility.
    a. This will present its own challenge:
        i. We will need to make sure that the timer is removed from the component tree once it has been done. We will need to do that ourselves as Flame will not do this for a repeating timer.
3. We will add a simple sound effect at each bounce, by adding a sound efx to our assets.
    a. **NOTE**: when two balls collide you want to just generate a single sound. Since collision detection will generate a notification for the pair (i.e. onCollision will be called twice for each collision)
        i. First, for object1 and object2
        ii. Then for object2 and object1
    b. We want to make sure that we generate the sound **only once** for each pair, otherwise you will get a strange overlap of sounds which will sound very weird.

Solution:

1. We are simply building upon Exercise #1 above.
2. We will use a small font so that it will not distract from the actual objects bouncing, we will also add an ordinal number to each ball (to see the order of ball creation.)
3. So the actual text format will be:
    a. **#NNN - NNN%** where the **#NNN** is the ordinal number of the ball being generated (1st 2nd etc…) and the **- NNN%** is the life left. For example:
        i. **#002 - 30%**
        ii. The above tells us that this is the 2nd ball generated, and it has only 30% life left.
4. We will create a separate TextComponent class for this purpose:
    a. To format the numbers, we will use the internationalized Flutter formatter which we will need to add to the pubspec.yaml file under dependencies:

```
 intl: ^0.17.0
```

    b. We will format the the pieces of the text as follows:

```
var orindalformatter = intl.NumberFormat("000", "en_US");
var healthDataformatter = intl.NumberFormat("000", "en_US");
```

    You can find more information on the formatter here: NumberFormat

    c. Once we have this we can then create the class as follows:

```
class LifeBarText extends TextComponent {
  final TextPaint textBallStats = TextPaint(
    style: const TextStyle(color: Colors.red, fontSize: 10),
  );
  var ordinalformatter = intl.NumberFormat("000", "en_US");
  var healthDataformatter = intl.NumberFormat("000", "en_US");

  /// text data we are formatting
  int _ordinalNumber = 0;
  int healthData = 0;

  /// Default constructor: gets the ordinal number of the ball
  LifeBarText(int ordinalNumber) {
    _ordinalNumber = ordinalNumber;
  }

  @override
  Future<void>? onLoad() {
    textRenderer = textBallStats;
    return super.onLoad();
  }
  @override
  void update(double dt) {
    text =
       '#${ordinalformatter.format(_ordinalNumber)}'
       '- ${healthDataformatter.format(healthData)}%';
    super.update(dt);
  }
}
```

5. To make sure that only a single sound is played for a pair of balls colliding, we need to ensure that the 1st notification generates a sound but not the 2nd.
   a. We track all collisions by hashing the ids of the two colliding objects as follows:
      i. Get the hash code of object1 and object2
      ii. Sort them in ascending order.
      iii. Create a key from the numbers by combining the two codes (as strings) and add the key to a set of such keys. We will keep a global variable for this. We then do the following:
         1. Check if this key already exists in our global set
         2. If NOT
            a. Add the key to a set of such keys.
            b. Play the bounce sound
         3. If YES (this means its the 2nd notification)
            a. Remove the key from the set
            b. Do not play the bounce sound.

You will find the code solution in the directory *“Exercise Solutions\component_005[Timers]\component_005_002”*