One Cord Hero

◎ Work tips

- the point is to do personal work, even if you work together and discuss
- all useful notions will be given in class or in the document
- read the questions, most the answers are in them
- ask questions 😄

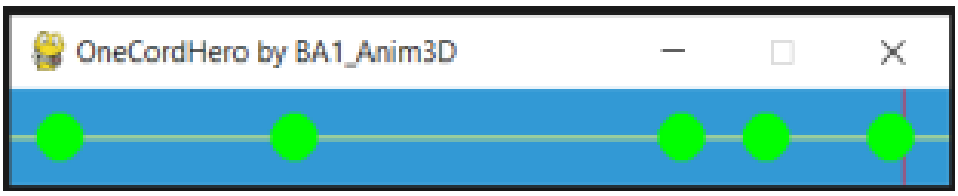
Context

The point of this project is to build a simplified Guitar Hero game step by step. We will simplify the game so that it only uses one guitar cord.

At any point, you can ask questions either in class or by email. You can also send your file without questions, and it will be reviewed.

Idea

The main idea in the Guitar Hero game is that notes have to be played when a symbol reaches a the line.
We are using a single cord guitar, and any key strike will allow to “play” the note.



Quick definitions

A **position** in the game zone is defined by 2 numbers that indicate the distance to the top-left corner (0, 0).

A **circle** is defined by the position of its centre and its radius.

At each game **loop**, the centre of the circle will move by 1 pixel along the width of the window. Its height will not change.

All the different **elements** (background, cord, notes, game line) will have their own style (colour etc).

The speed of all the notes will be the same.

Template discovery

You are provided with a code **template** that you have to fill in to be able to play.

The file is named **`one_cord_hero_template.py`**. Download this file[1] and replace "template" by your last name.

Step 1. Variable declaration

As with all the projects you will do, declaring the initial variables is a good way to get started.

The variables we are goin to declare allow us to define the playing area and make sure the dimensions are proportional. We will base this on the radius of the circle.

In the file, define the variables to manipulate the following values:

- the **base radius** of a circle, with value `10`
- the **width** of the game area, equal to `40` radii
- the **height** of the game area, equal to `4` radii
- the **position** of the **line**, which should be the width of the window minus `2` radii
- the **position** of the **cord**, which should be in the middle of the game area
- the **colour** for the notes
- the **colour** for the cord
- the **colour** for the game line
- the background **colour**

Step 2. Window preparation

We now have to create the context in which the window will appear.

Define a variable that will contain the **window** object by using the `width` and `height` variables you have previously defined.

> **Idea**
>
> The gaming area must change is you update the base radius value!

Fill in the window's background.

Useful functions are defined below!

[1] from here: `https://irwin.sh/documents/teaching/rubika/scripts/one_cord_hero_template.py`

</> Code

```
# Stores in a variable a window with width and height
window = pygame.display.set_mode((width, height))
# Fill a window with a given colour
window.fill(colour)
```

Step 3. First graphical element

In this step, we will write the instructions to draw the guitar cord and the game line.

For each of these, you should use the variables you have previously defined. This will allow the lines to stay correctly positioned event if the game area size is updated.

Useful code snippet incoming!

</> Code

```
# draws a line
pygame.draw.line(window, # in the window
          colour, # in that colour
          (x_st, y_st), # from this point
          (x_end, y_end), # to this point
          width) # with that width
```

Step 4. Variables for the game

These new variables will allow to store the data useful for the game.

You will need:

- the **number** of **notes** (start by using `10`)
- a list to contain the **positions** of the **notes**
- a **score** (start at `0` of course)
- a **clock** measure (start at `10` for the moment)

Step 5. Game main loop

This step will create the game's main loop, also known as the code the game will have to execute until it's finished.

Idea

The game is finished when there is no more notes to show, also meaning the position list is empty

What you need to do:

1. change the main loop's stop condition
2. correctly call the `game_turn` function by correctly using the parameters and returning the correct values
3. the clock should use the measure you created
4. update the end-of-game message to display the player's score

Idea

The idea for the loop is: "while there are notes to display; update the score and the notes' position with the game turn function update the clock"

Step 6. Defining the distance

We have to define the distance between two notes, as to show them nicely on the screen.

The distance we will start with will be chosen randomly between :

- 2 radii
- half of the window's width

Idea

For this, you have to import the `randint` function from the `random` package.

Define the `get_random` function to do this.

Step 7. Showing the notes

We have defined a way to compute the distance between two notes. We now have to show them on the screen.

For this, define the following functions:

- `draw_note`
- `draw_note_list`

The second function is used to draw **all** the notes. For that, you probably want to loop the note list and call another function.

Step 8. Defining the notes

We have to find a way to compute the initial positions for the notes. Our first version will use random positions.

Write the body of the `get_random_positions` function so that it takes one parameter. This parameter is a number. The function will return a list of that size with the notes' positions. The notes will be separated by a random distance.

The function should:

1. create an empty list
2. define the first note's position to `0`
3. **for** the asked number of times:
   (a) choose a random number
   (b) substract that number from the last position
   (c) add this new position to the list
4. **return** the list

Step 9. Moving the notes

At last has come the time to move the notes forward[2].

To do this, write the body of the `scroll` function. This functino takes the position list as a parameter and returns an updated version.

To update the positions, add `1` to every position in the list.

If the first position is out of the window, we remove it from the list. We can then return the updated list.

Step 10. Seeing the notes move

- Add a **call** to the function that draws the notes (see Step 7).
- Add a **call** to the function that updates the position list (see Step 9).
- Before the game starts, update the position list you created by **calling** the function you created at Step 8.

Step 11. Playing a note

We consider that a note is **played** (and that we earn `1` point) if the centre of the note est at less than a radius from the **game line**.

[2] if we don't, the game becomes very boring...

Write the body of the `play_note` function. This functions takes as parameters the list of the notes' positions and the current score and returns them both updated.

You only test the **first** position in the list. If the note's position is deemed correct, then the note is removed and the score is increased.

Step 12. Reacting to an event

We want to be able to react to user events. Add the code that allows:

1. to handle exiting the game window (clicking the × implies the game is over)
2. to handle a keypress (then we play a note)

Idea

The event sent by clicking the × is `pygame.QUIT`. The event sent by a press on the keyboard is `pygame.KEYDOWN`.

Step 13. Extensions

1. Add a test in the function you wrote in Step 9. If the last position of the list is greater than `0`, then we add new positions.
2. Update the `game_turn` function to accelerate the clock every 10 points
3. Count the number of missed notes. You can use this to give a limited number of possible missed notes to the player.
4. Try adding a cord ?
5. Update the game art ?
6. Add music ?