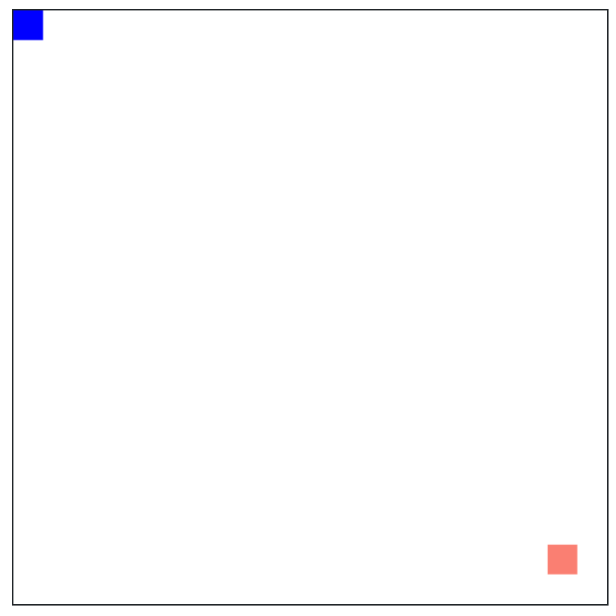
**Baby Snake**

In this assignment we are going to make a baby version of the classic Atari game of [Snake](https://en.wikipedia.org/wiki/Snake_(video_game_genre)). It was famously shipped on the original Apple II computers as well as Nokia phones. Here is a demo with a few extensions: [Baby Snake Demo](https://codeinplace.stanford.edu/cip3/share/gy05Az3r7HyrDK7t8oUx)

**Milestone #1: Set up the World**

Draw the following world! It has a blue square is the "**player**" the red square is the "**goal**". The player and the goal are both 20 pixels by 20 pixels.

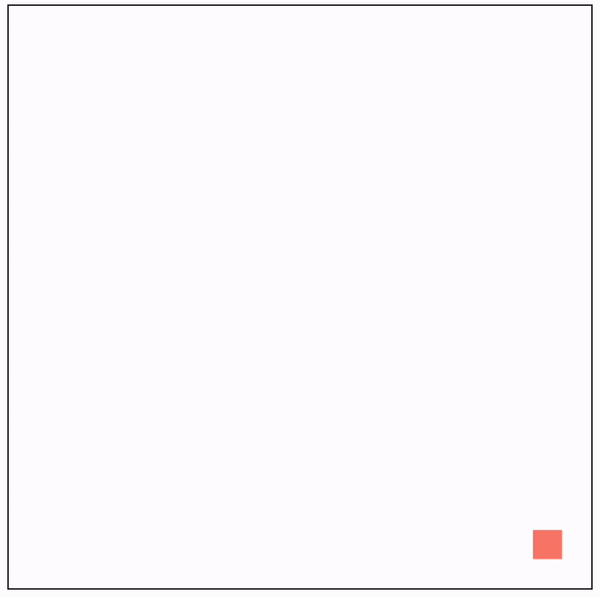
The player should start in the top left corner of the world. You can place the goal anywhere you like **as long as** its x and y values are both [multiplies of 20](https://www.cuemath.com/numbers/multiples-of-20/). Here is a reasonable configuration where the goal is at (360, 360):



Keep in mind that later on, your animation loop will need to access the player and the goal variables (the values returned when you call create\_rectangle). Speaking of which, it is time to animate!

**Milestone #2: Animate**

Each time through the animation loop you should move your player 20 pixels (this size of the player) in the direction it is traveling. The directions are either left, right, up or down. At the start the player should be traveling to the right:



*A gif of the animation (which loops)*

Recall the standard animation loop:

A screenshot of a computer program

AI-generated content may be incorrect.

Recall that you can move an object by using either of these graphics functions

A close-up of a white background

AI-generated content may be incorrect.

Once you have completed this milestone your program only animates the player moving right.

**Milestone #3: Handle Key Press**

the direction that the player is traveling can either be Left, Right, Up or Down and should be controlled by the keyboard.

We haven't seen the keyboard in Code in Place. It works just like the mouse! At any point in a graphics program you can ask the canvas for the last key press

A black text on a white background

AI-generated content may be incorrect.

The key variable will then hold the *name* of the key which was last pressed (or None if no key has been pressed). If the name of the key is "ArrowLeft" that means the last key the user pressed was the left arrow. Here is an example of printing arrow keys:

A screenshot of a computer code

AI-generated content may be incorrect.

You should process canvas.get\_last\_key\_press() inside the animation loop. We strongly encourage you to keep a variable which stores the current direction of travel.

**Milestone #4: Detecting collisions**

If the player goes out of bounds, the game is over. Write code that checks for, and handles, out of bounds cases. You will likely benefit from using:

A close-up of a number

AI-generated content may be incorrect.

where the obj parameter can be either the player variable, or the goal variable. The return type is always an integer, so you can compare the value of say the player x and the goal x.

**Milestone #5: Moving the goal**

When the player hits the goal, you should randomly move the goal to a new location, anywhere on the board. Write code that detects if the user has hit the goal, and randomly places the goal in a new location. You will need to use the randint function which returns an integer in the given range:

A black text on a white background

AI-generated content may be incorrect.

Recall that the new x and y of the goal should be multiples of 20, otherwise it may be hard to detect if the player and the goal touch. There are many algorithms which you could come up with to get a random value which is a multiple of 20. Also recall that randint is inclusive, which means it can return 0 and it can also return max\_value.

**Extensions**

You did it! Congratulations, that was a huge program. If you want to keep going, awesome. Add any extension you like. Here are a few that we think are fun:

1. Get faster each time the player touches the goal.
2. Keep track of points
3. Add obstacles

**Full Snake?**

If you are feeling very adventurous you could try and implement the full game of snake. Here is a link to an online version: <https://playsnake.org/>. The full game of snake is a hard challenge even for a well seasoned programmer. If you do go down this path, here are a few pro tips:

* Program your snake in a new project (leave this one as baby snake)
* Represent your snake using a list of rectangles (where the rectangles are the shapes returned by create\_rect). This will make it much easier to move your snake. You will only need to change the head and the tail.
* Us the find\_overlapping function to tell if you have hit yourself.