Coding Club Meeting 2

Goals for Meeting

- Get more familiar with some of the functions in scratch, including if and while statements, sprite cloning, variables and lists, custom blocks, and sensing and operator blocks by creating a basic game

- Learn what a game loop is and how we can create one

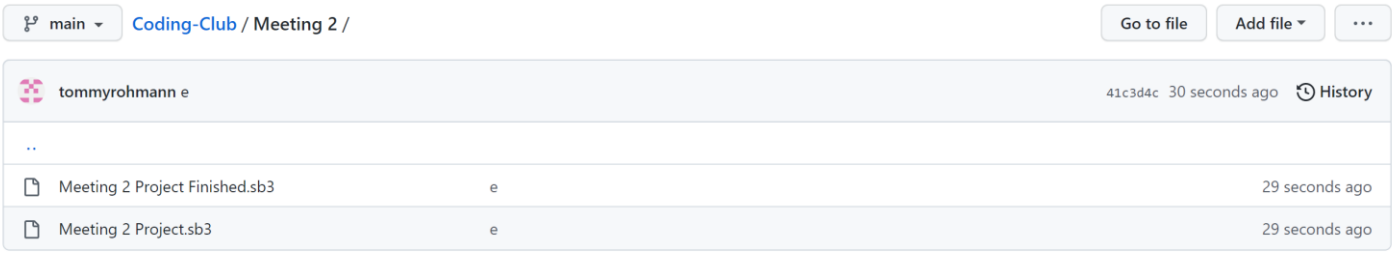
Our game will have:
- Start Menu
- A player and enemy
- Losing Screen
- Score
- Leaderboard

Getting Started

Lets start by getting the meeting 2 project from the coding club GitHub Page

https://github.com/tommyrohmann/Coding-Club

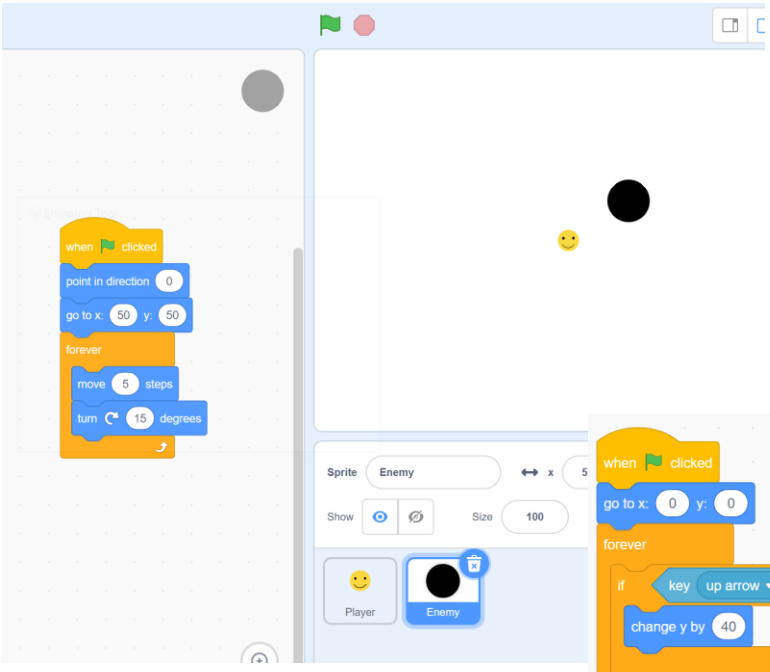
Download Meeting 2 Project.sb3 through the path shown below.



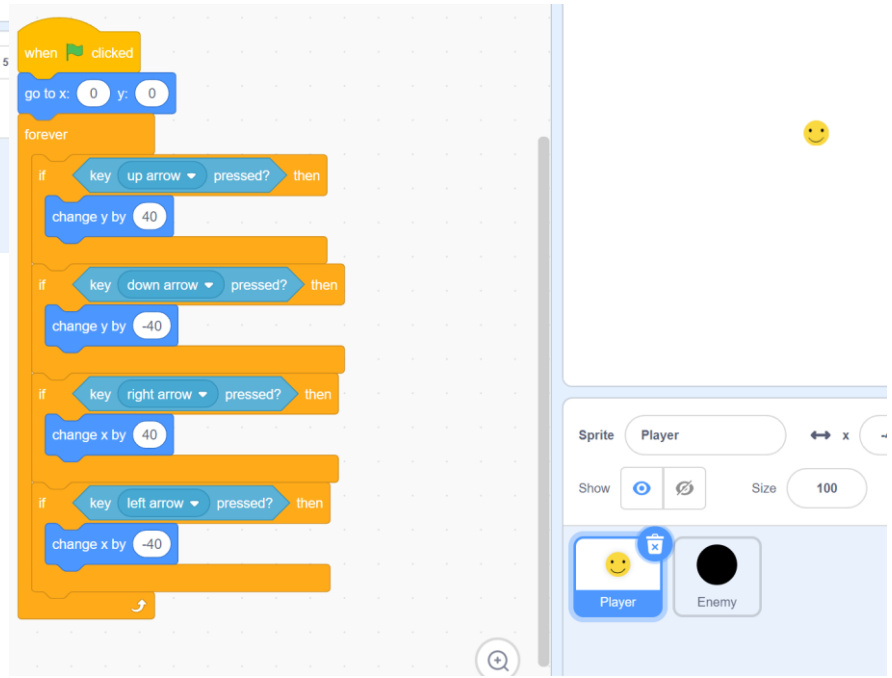
Open the downloaded file on scratch. This will be our starting point for the project today.

What We're Starting With

Looking at the project we jus downloaded, it has 2 sprites, a smiley faced "Player", and a black dot for an "enemy". The black dot has a code that says when the green flag is clicked, it will go to an initial position, and then move in a circle by rotating, and then moving a small number of steps in a loop forever. The player is a little more complex, with a loop that uses "if statements" to check if a player is pressing one of four arrow keys on their keyboard. If an arrow key is pressed, the code will detect which one, and move the player in a corresponding direction.



Enemy



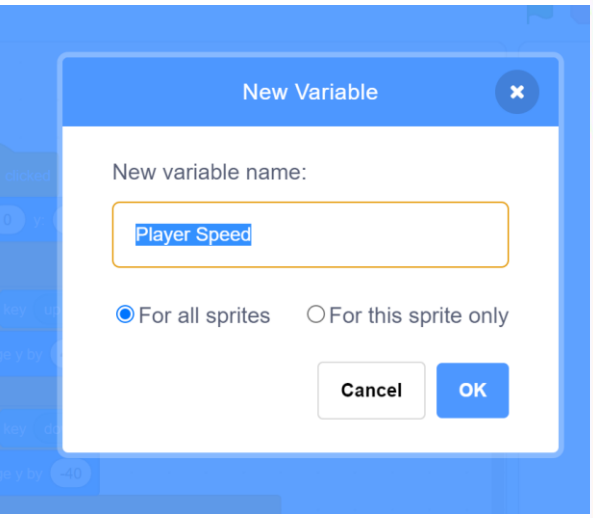
Player

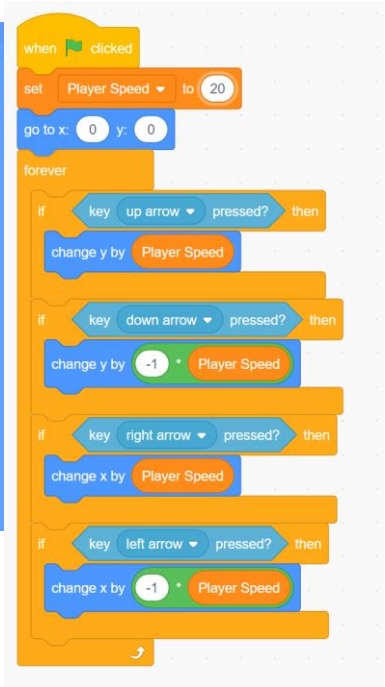
Improving Movement with Variables

Lets see what everything is doing by running the code. Click the green flag, you might notice that while you can move around, the player moves too fast. While we can go ahead and change the numbers manually in the script, we can also use a variable to store how fast we want the player to move. This way we can change all four numbers at once, or even change how fast the player moves in code while the game is running.

To get the player to move in the negative direction, the multiplication operator block can be used to multiply the velocity by -1, making it so the same variable can be used to move the player in the opposite direction.

Create a new variable and call it “Player Speed” For now, when we make variables, check “For all sprites”. Lets add some code to set the variable when the green flag is first pressed, and then update the code as follows:



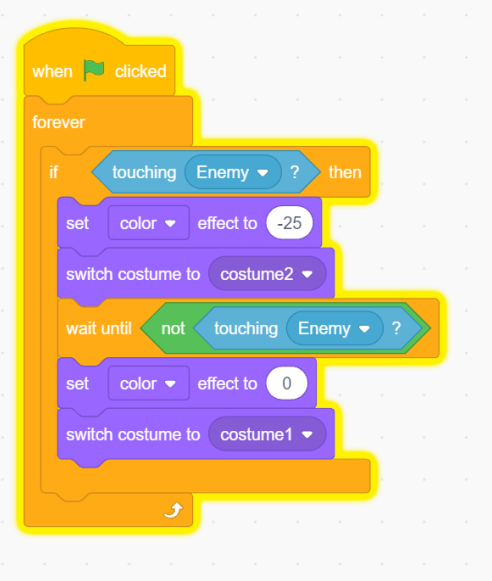


Shown below, changing the speed to 20 is still too fast, but fortunately, changing the speed again is as simple as changing 1 number in the code

Detecting Enemy Collisions

Now we have some movement in our game, but there really isn't much to do in it. Lets just try making it so the player gets hurt when they run into the enemy.

We can do this by making another script. Using a sensing block, lets detect when the player is touching the enemy and have this code trigger some sort of indication that there was a collision. Using the "Looks" Blocks, the sprite color and costume, which is the picture displayed by the sprite, can be adjusted. Lets make a code to have the player look hurt, and then change back to being happy after not touching the enemy anymore.

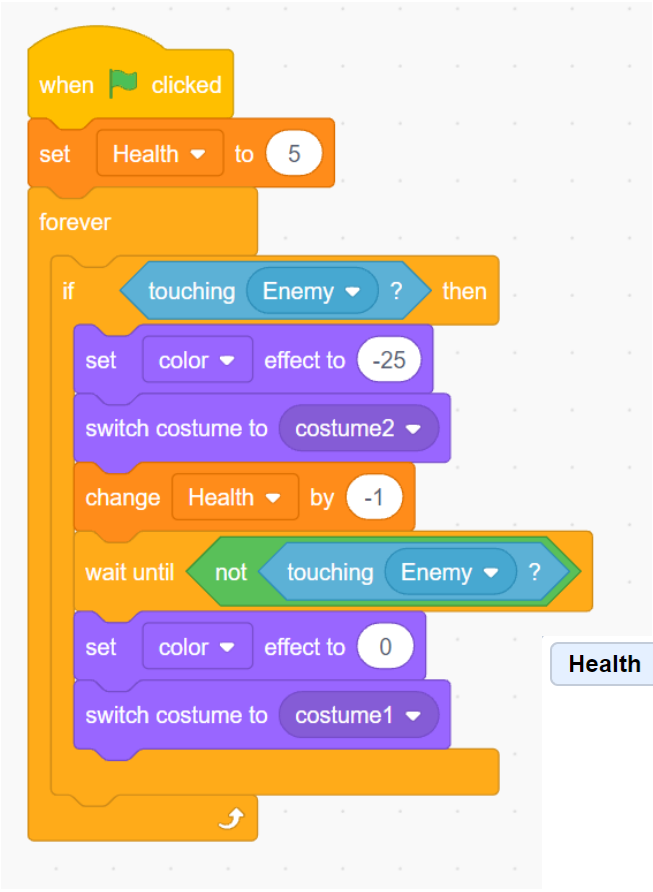


We can analyze what the code by reading what occurs in the order it is executed in. When the green flag is pressed, a forever loop will initiate. Here, if touching the enemy, the player will change costume and color and then wait until no longer touching the enemy. Then the player will return to its original state. If not touching the enemy, the forever loop will run back and check for a collision.

Detecting Enemy Collisions

Lets also track how many times the player has been hit by the enemy by establishing a “Health” variable. Remember, variables don’t necessarily do anything on their own, but we can alter it and use its value to give it meaning.

With this variable, lets have it be set to some starting number, say 5 when the game starts. Lets also add to our collision code to make the player lose 1 health whenever they run into the enemy.

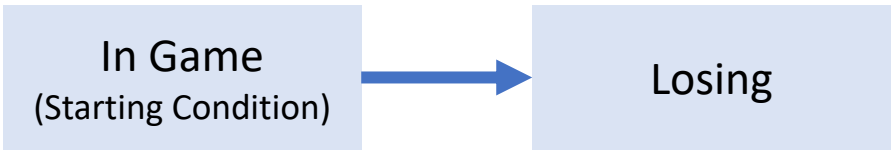


Great! We can now track the players health by effectively counting how many times they have been hit. That said, we still have the same problem as before, this health variable doesn’t really mean anything yet as shown if we get hit more than 5 times. To improve on this code, lets talk about game loops.

What is a Game Loop?

While its great that we can hit the enemy and make our player look hurt by visual cues, and we added a variable to track the players health, this collision doesn't necessarily mean anything yet. Lets try to establish a game loop.

A game loop is how your game will be structured and is the general path of screens and modes a player will take as they play their game. For this game, lets try to make a game loop with 2 states:

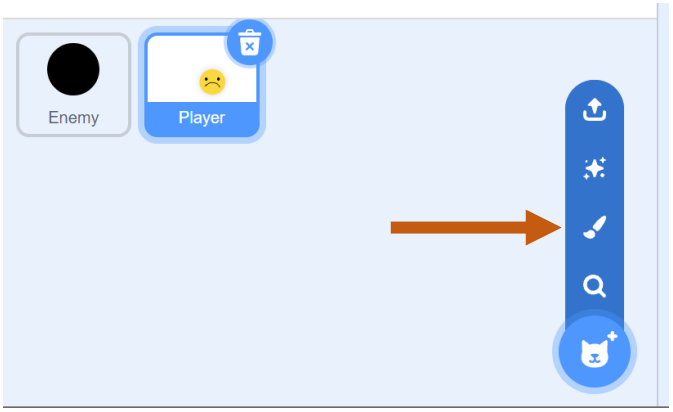


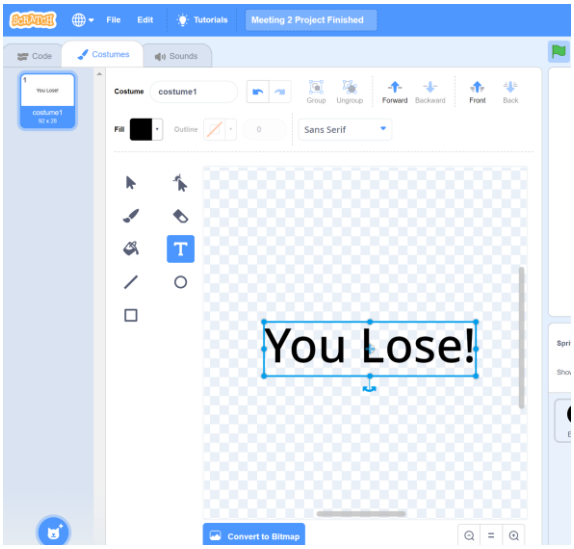
All of this may seem rather arbitrary and vague, and that's because it is more or less. Really none of what function added to the game has any meaning, so we need to create effects that add meaning to each of the functions in game. In this case, lets just make the game stop when the player reaches 0 health.

While we could just insert the block that simply halts our code, lets try to instead hide all of our sprites and have the game display a message saying that they lost.

Making a losing screen

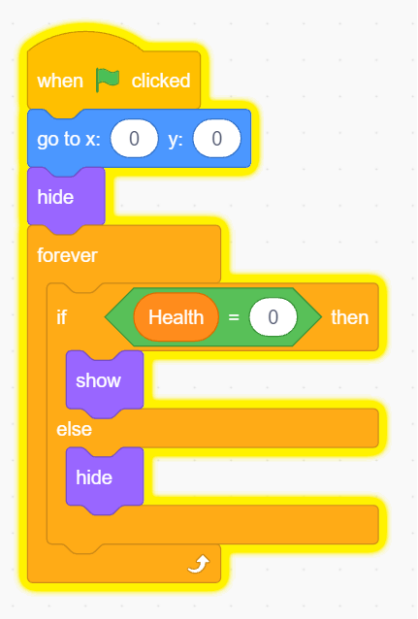
To make the losing screen, lets make a new sprite and using the text tool in the costume tab, lets just type “You lose”. Make sure its centered in the middle of the canvas.





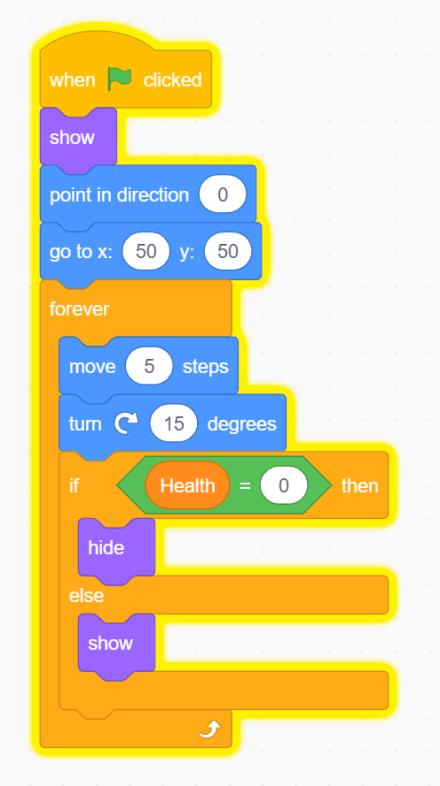
With the sprite made, lets make some code for it. We want for the sprite to be hidden, and then when the losing condition (the player losing all their health), the sprite will show.

Of course we also need for all of the game pieces (being the player and enemy) to become hidden as well, so we will need to append the code here as well.

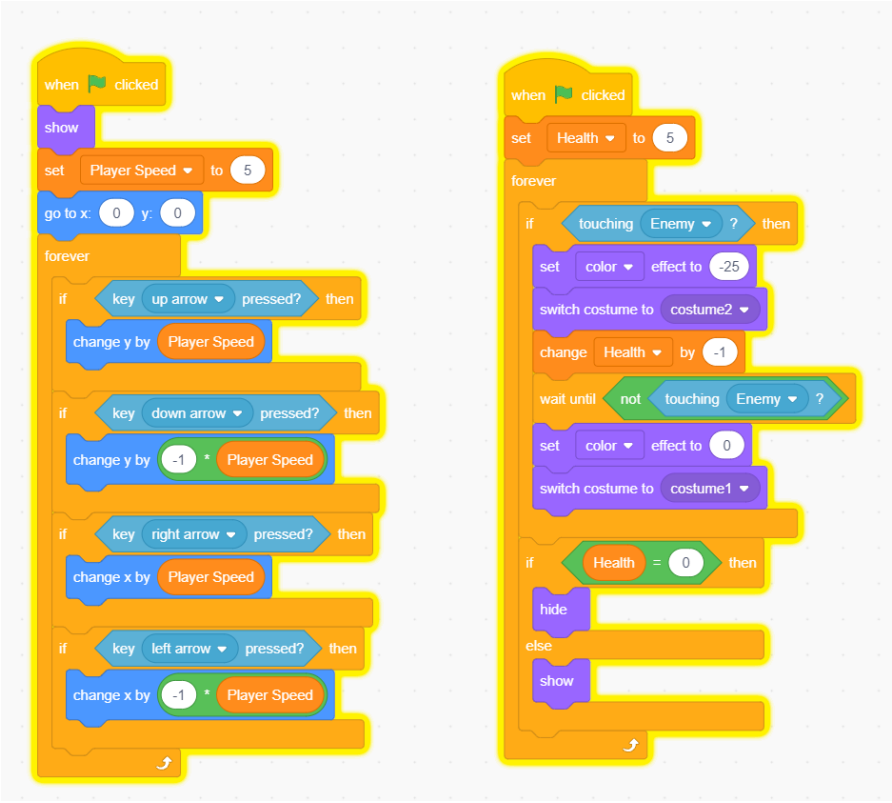


Making a losing screen

Enemy Code



Player Code

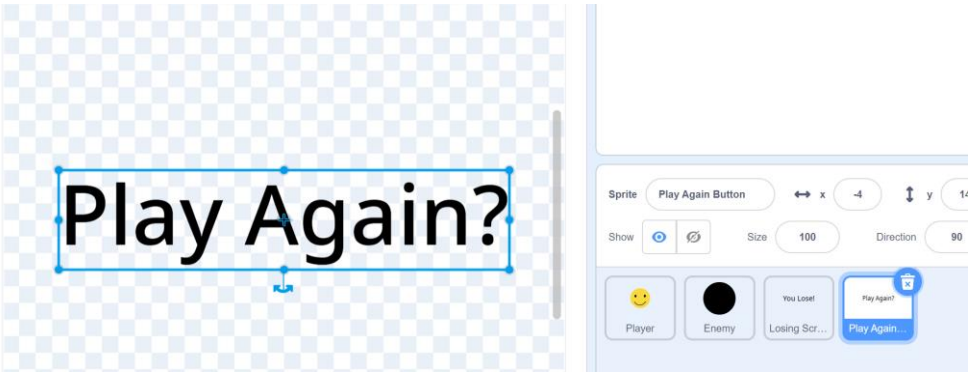


Here, we use if else statements so that this condition can also be used for the reverse, say if health were to be restored after losing the game.

Restarting the game

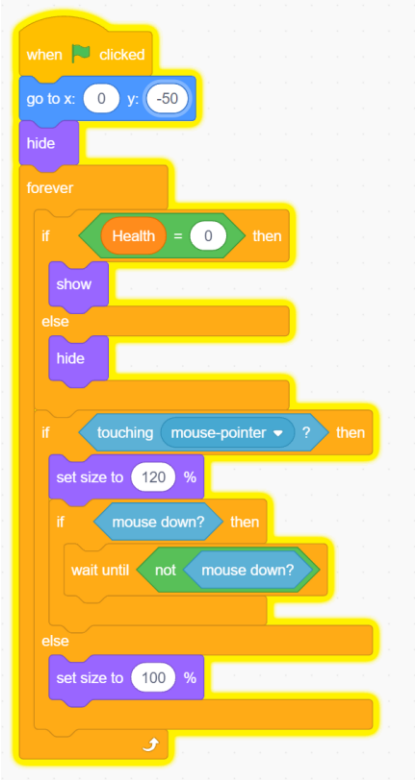
With this done, we now have a way for the player to lose, but our game loop is not too much of a loop right now. Lets make a way for the game to start again if the player wants to play again. To do this, lets make a button also appear on the losing screen.

So far, the base code we want is the same as the losing screen button, so lets just copy the losing message sprite and edit its costume.



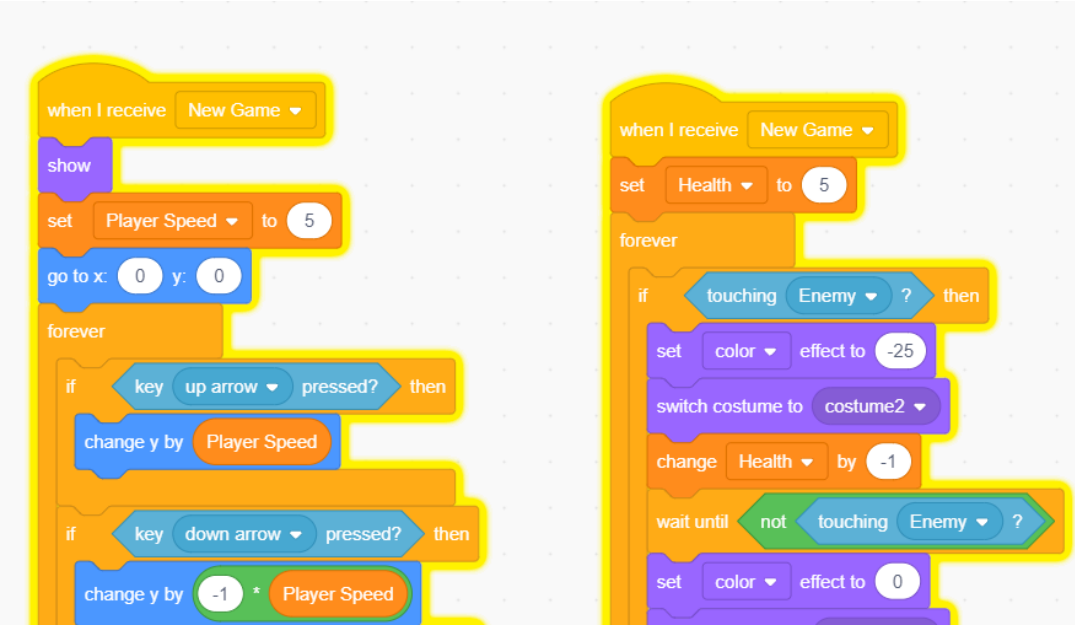
From here lets edit the code. We want for the player to click the “play again” button, for this, we need to detect when the mouse is touching the button, and when the mouse is down. We can use an “and” Boolean to detect when both conditions are true, but we might also want additional functionality, such as for the button to enlarge when touching the mouse to indicate that the user can interact with it. Taking what we learned from making the other sprites, lets make a code for this and make some adjustments to make the button look nice on the screen.

Restarting the game



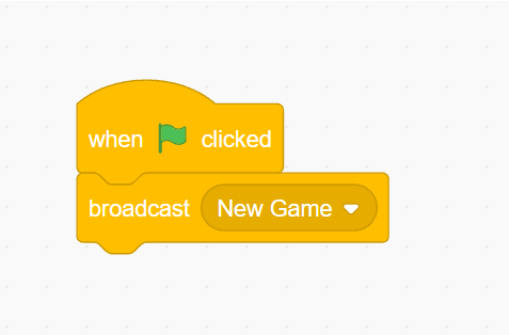
Lets have the code look like this for now. From here, we need for this button to trigger the game to restart when clicked. For this functionality, lets try using a signal block. The signal block works by announcing an event. This event will trigger something happening. Lets put an announcement block and have the button announce “New Game”.

Now, just like everything else, we need to edit our code to have the announcement do something. Lets just change all of the game pieces so that instead of having their code start when the green flag is pressed, lets make them start when they receive this announcement.



Starting the game

What we just did created an issue however. Now, the game has no way of starting when the green flag is clicked, which is usually how a code would initially be started. Lets just make a code that just announces new game when the green flag is clicked. Lets make a new sprite for this.

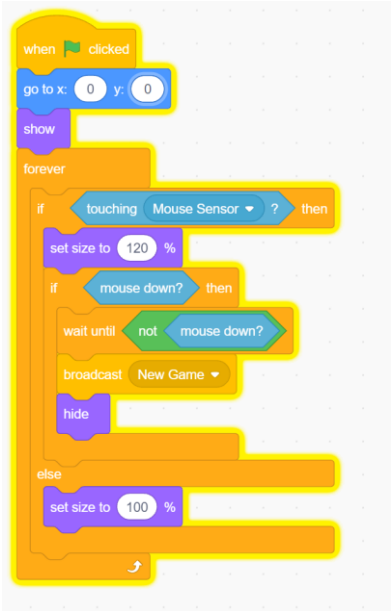


For now this will work, but if you want to, try and apply what we have learned so far, try making this sprite a button that starts the game initially, like adding a start menu to our game loop.

Now, we should be able to run our game with a losing screen, restart button, and maybe even a start button, but there are a few issues we need to iron out first…

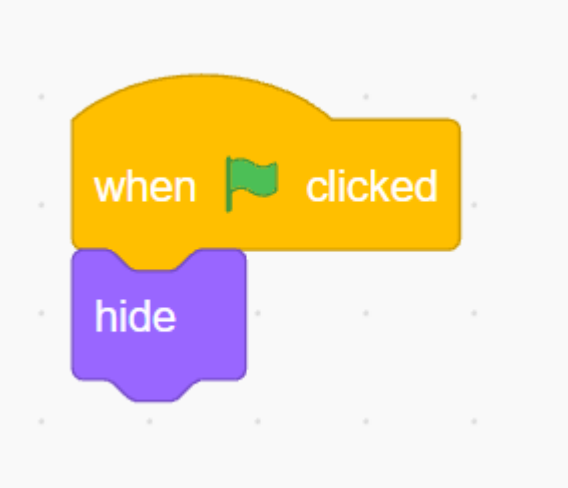
Making a Start Screen

Trying to make a start button, you may have come up with a code that looks like this:



If not, it is ok. If your button code did not work, the one above will work just fine.

Since we are making a start menu, we do not want our sprites to be showing when the game is first starting. While they can be in a hidden state, this is not always going to be true, so lets add a quick little code to all our game pieces saying to hide when the green flag is pressed.



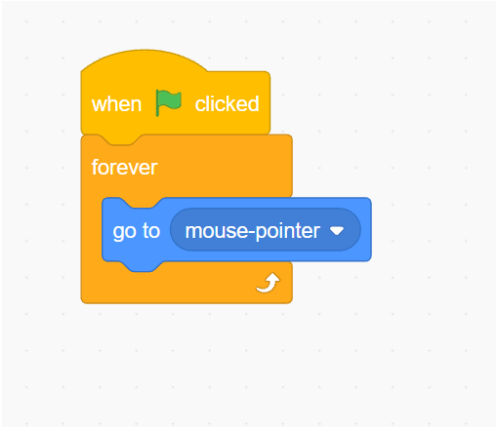
Making Good Buttons

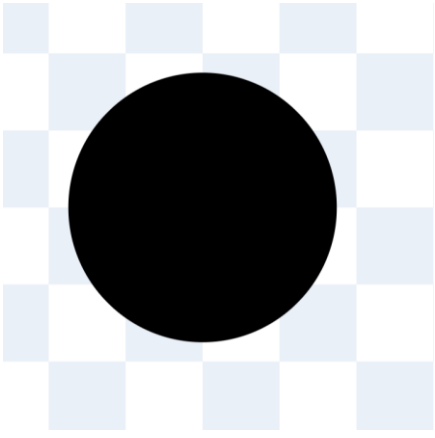
When clicking these buttons, you might realize its rather challenging to do, and the indicator we made may be constantly toggling. This is because of how sprites work. Since the costume created has transparent parts on it, those locations on the sprite are not considerer apart of the sprite, meaning if the mouse is touching a space in between letters, the mouse is not touching the button. We could edit the sprite to have a white box behind it, but if we wanted to use a background, this would cover it.

Empty space not apart of sprite

One solution is to make a new sprite that follows the mouse around, and instead of having the buttons detect if touching the mouse, they can detect the sprite, which we can make as large or small as we want. Another solution is to create an additional sprite for each button that will act as a collider for the mouse, but then a 2nd sprite has to be made for each button we add.

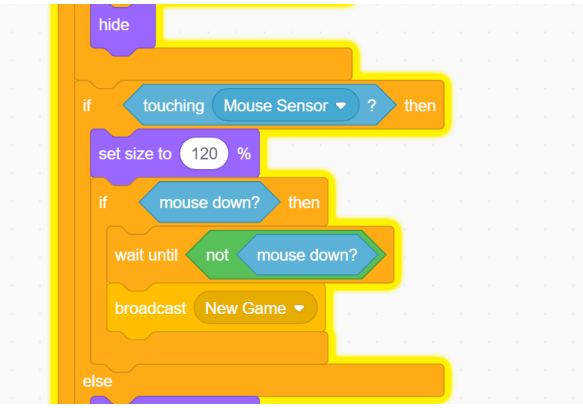
Lets just make this new sprite a small circle that follows the mouse around. We want this function to carry out independently from the game loop, so lets just put this in a forever loop that starts when the green flag is clicked.



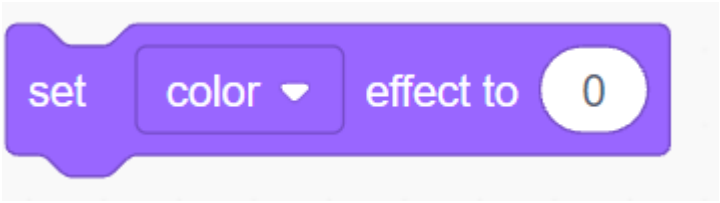


Making Good Buttons

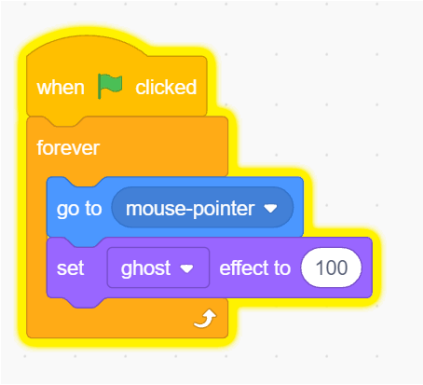
And of course now we need to update the code on our buttons to reflect this change:



This solution works great, but now we have a black dot on the screen all the time, so lets talk about how to hide it and keep the functionality. We could use the hide command, but this function actually also turns off sensing from the given sprite, making this block very useful for other applications, but not here. Also in the looks tab, you can find this block:

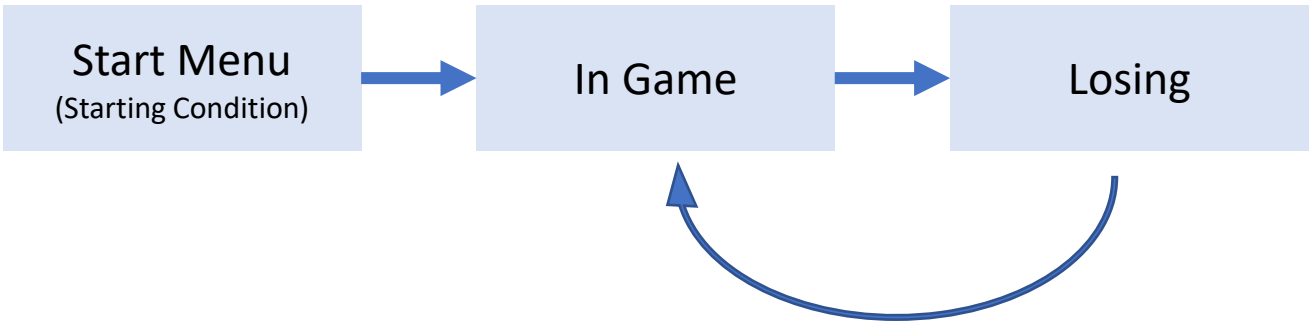


While we can set the color effect, there are also additional options that can be selected by clicking on the dropdown menu. By changing transparency (Called the Ghost Effect), we can make our mouse sensor completely transparent but still detectable by other blocks



What We Have So Far

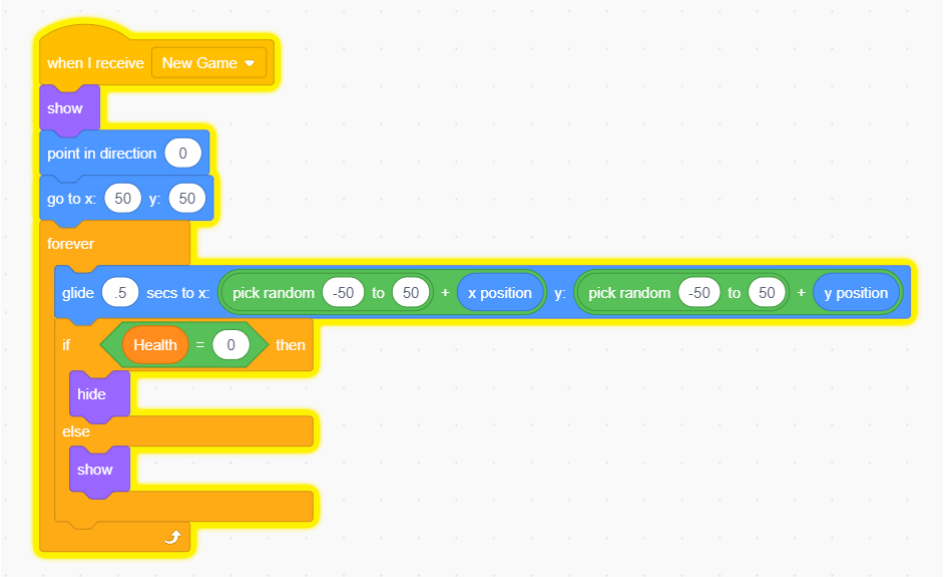
Up until now, we have just structured our game to fit into a planned game loop, which if we mapped it out visually would look something like this with arrows indicating how we can travel between states:



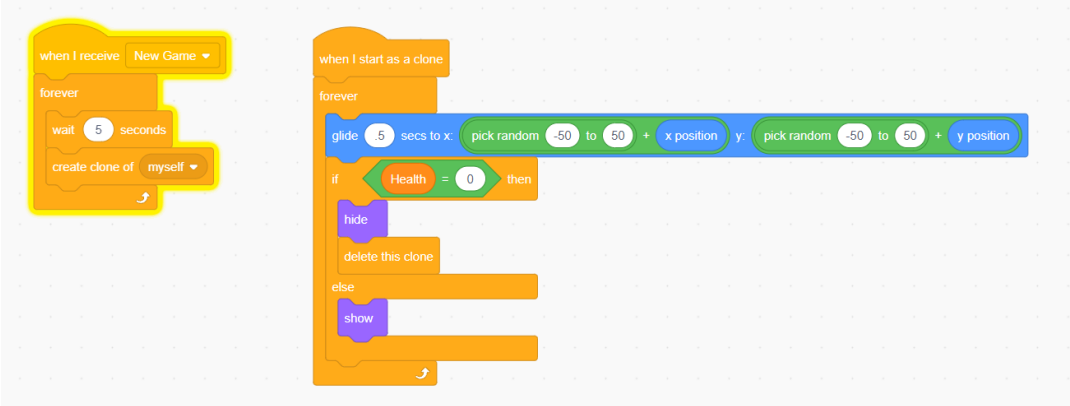
As you can see, there can be a lot of moving parts in structuring the game loop, especially as they get more and more complicated. In the future we will work on modularizing this system to make it easy to track and edit, but for now the actual game we have made is not very fun, so lets work on that. For now.

Improving Our Game

From earlier we made our player capable of moving, but the enemy we made the player to avoid is a little to easy to dodge, so lets make the enemy more interesting and challenging to deal with. To keep it simple, lets just say we want our character to move around randomly, so lets add this to our code:

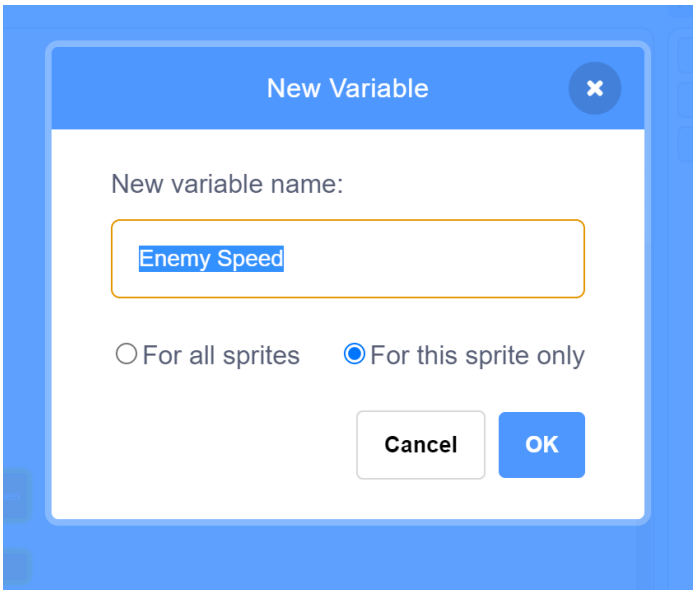


This still leaves us with a game so easy that its boring, so lets make it so that the game gets harder as we progress. We can do this in a variety ways, but how about for now we will try out the clone feature. The clone feature makes a clone of a sprite. Through this we can have multiple enemies without having to manually create them. Lets make a code that clones the enemy every few seconds. We will delete these clones once the player loses.

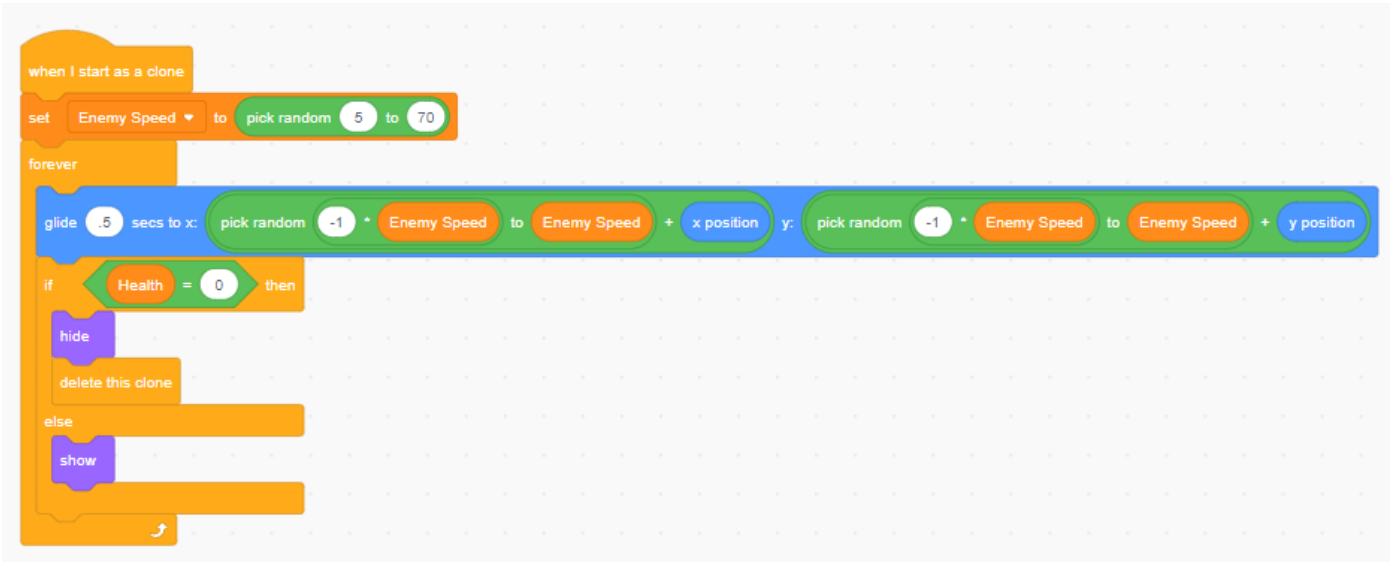


Improving Our Game

We can improve our game more by adding some individuality to our clones. Lets make a new variable and call it “Enemy Speed”, but this time check “For this Sprite Only”. It is important that you make this variable in the enemy sprite, otherwise the enemy sprite will not be able to utilize the variable.



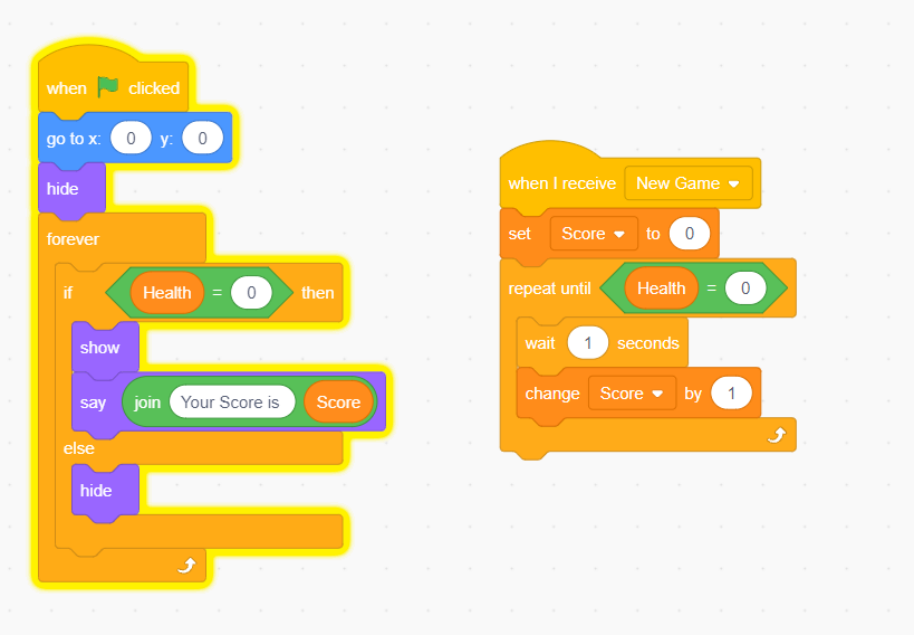
Now the variable is pertaining to this sprite only, and does not exist outside of the code of that specific sprite. This also applies to clones, so now, we can edit our code for the clones so that each one has a different variable. Lets use this to adjust how much each sprite moves every time the code loops.



Adding a Score System to Our Game

With this simple addition to the game, we made it rather challenging as time goes on. To finish with our game, lets add a score system. To do this, lets just make a new variable that is reset when we receive a start game message and counts up as long as the player does not lose. We can also have a sprite say the score when the game ends to tell the player how well they did.

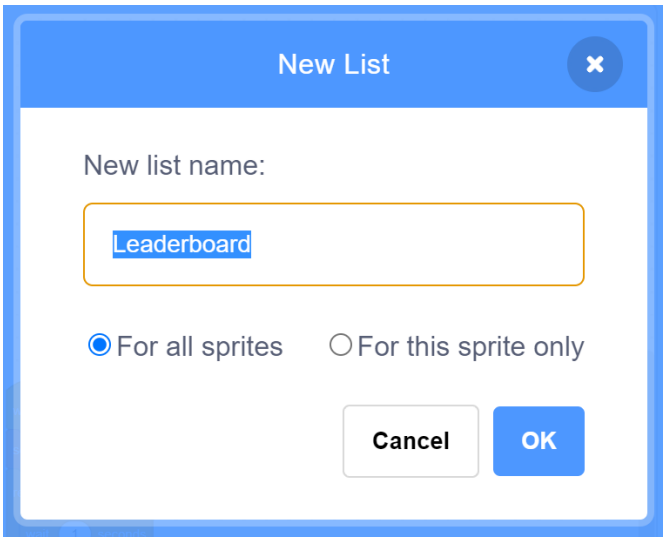
One way of doing this could be as is shown below on the Losing Screen Sprite:

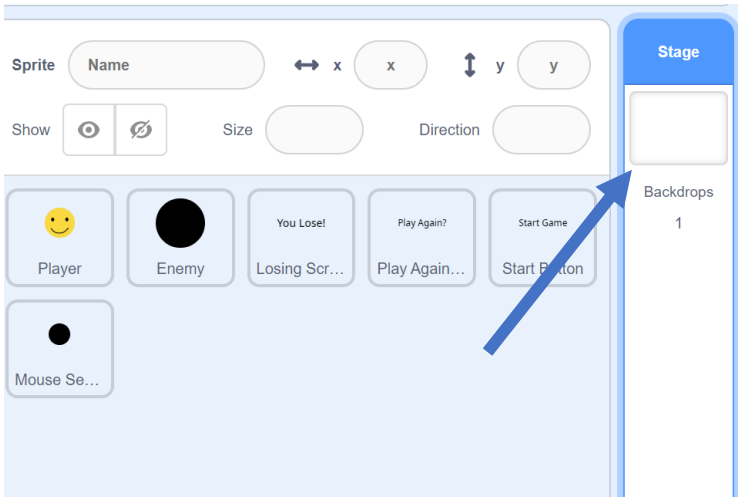


Creating a leaderboard using lists

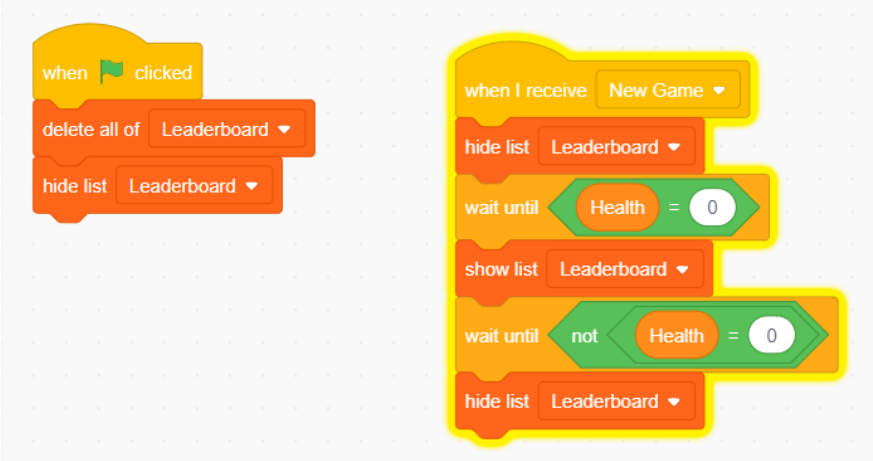
While scratch is entirely contained to the scope of the code being run once, meaning there is no way to save variables from the project without editing it, we can make use of lists to track scores in a given playthrough of the game.

Lets start by making a list, which is found underneath the variable section. Lets name this list “Leaderboard” and under the stage script, lets start making a code to control it



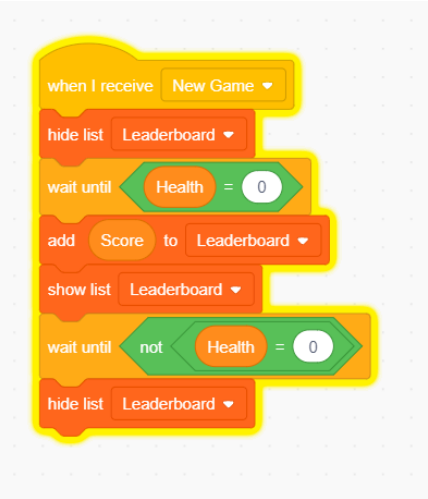


Lets start by making a code that will ensure the list is clear when the game is first started, and will not be showing initially, but when we get to the losing screen, the list will display all the collected scores. We also want the list to hide when the game is restarted.

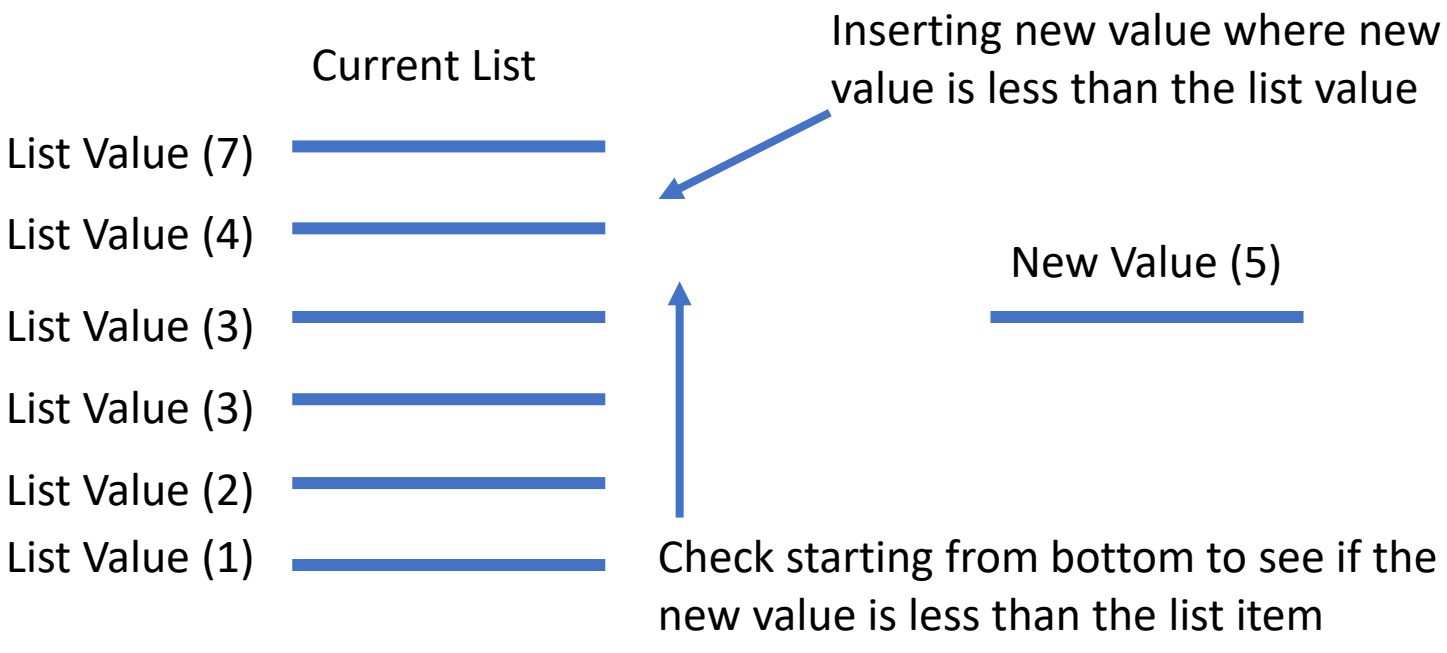


Creating a leaderboard using lists

Now that we have a list that hides and shows itself when we want, lets make the list store values. To add to lists we would use the “add item to list” block:

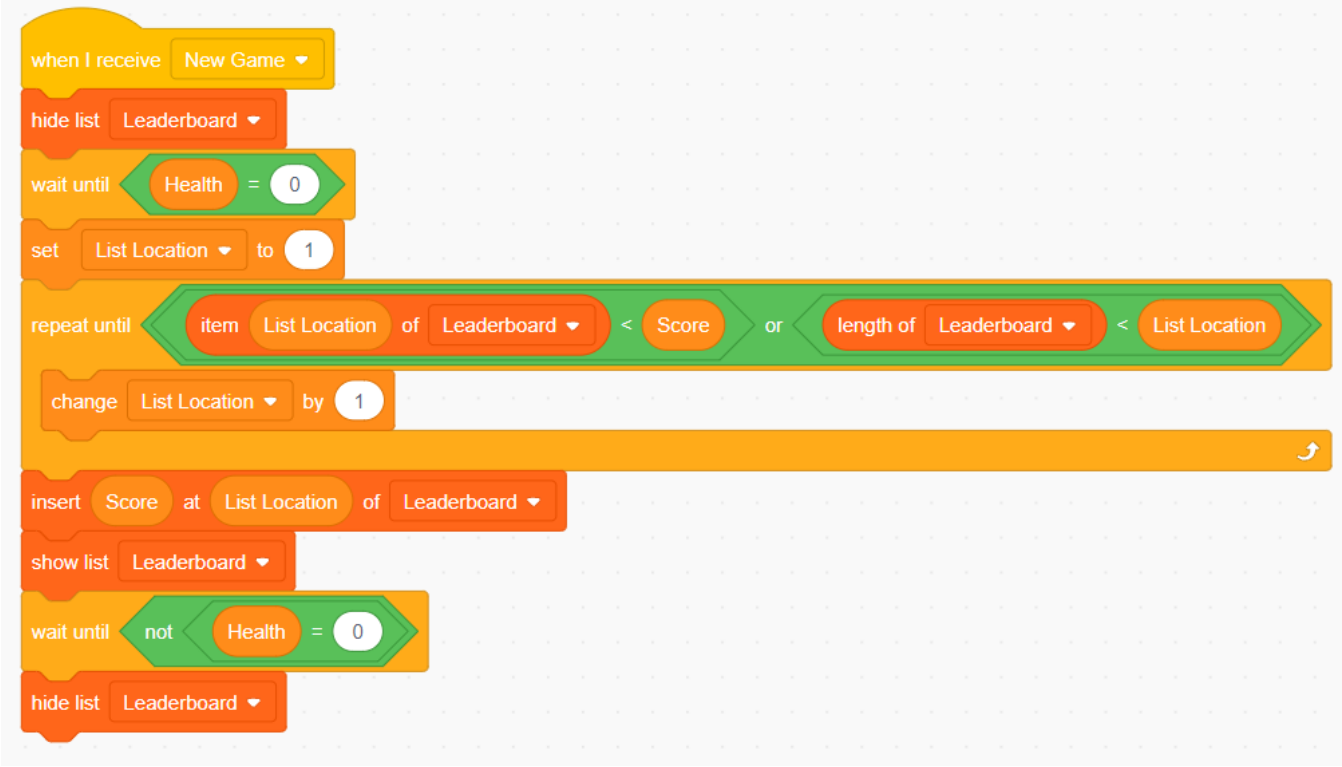


but we want to sort the order of the list by the order of scores received. To do this, we need go through every item of the list and compare values until we find where the score to be added should be located. We can do this looking from the bottom of the list up, or from the top down. To do this, we need to store a variable to count how far into the list we are, and then insert the value in the list at that point when it is found.



Creating a leaderboard using lists

A potential Code for this would work as follows, but there are many ways of doing this.

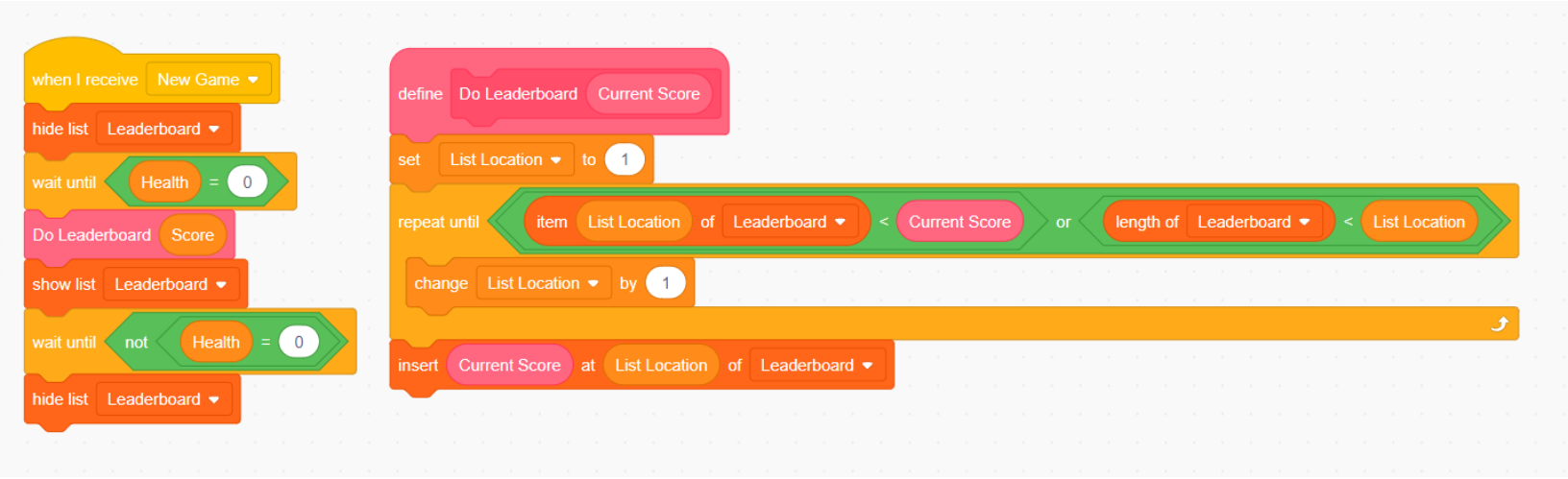


And just like that, we made a leaderboard.

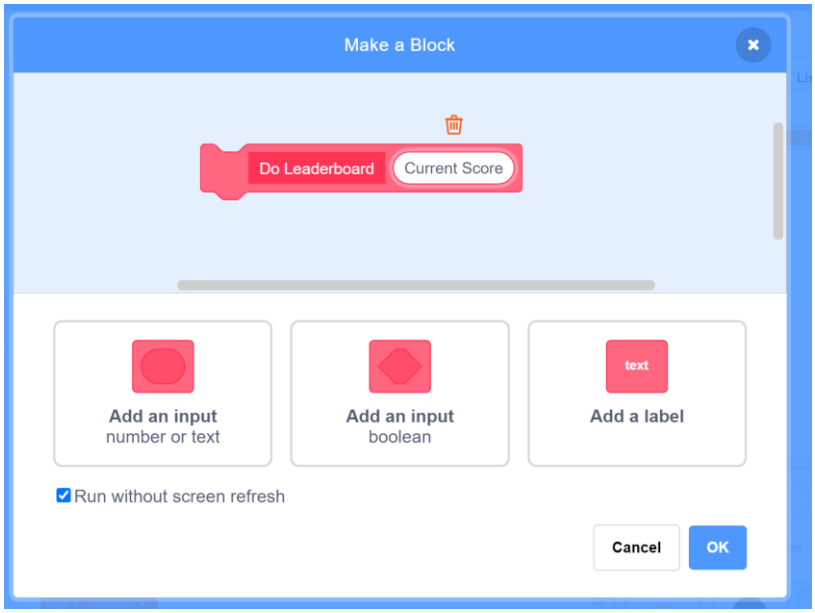
Now, this whole line of code does one function, so lets try using custom blocks to simplify it. A custom block takes a whole set of code, and condenses it into one block which you could place anywhere for a given sprite. These blocks also have inputs, so we can input score, but if we wanted to look at another variable, such as the number of times played, we could replace it in one location rather than everywhere in the code.

Simplifying Code with Custom Blocks

A potential Code for this would work as follows, but there are many ways of doing this.



Another great effect of this function is that you can run entire sections as one block of code, by checking “Run without screen refresh”, the block will run as though everything within it is carried out between frame updates. This is extremely useful for large or visually oriented portions of code, and is also used for stuff such as physics in games and more.



Bugs and Design Oversights

While this project is great for learning different blocks and what they do, it is not the greatest or cleanest running code. What bugs do you see? Are there any designed oversights that might lead to exploits from the player? Are all of these unintended attributes of our code necessarily bad? How can we fix these bugs?

Lets discuss it.

Uploading our games to the GitHub

Just like last time, we will be pushing our code to the GitHub page. This time we will be uploading an entire SB3 File (Scratch project). To do this, save your project somewhere on your computer, be sure to name it something that is not the original file name (Meeting 2 Code + your GitHub Username would be preferable).

From here navigate to the meeting 2 folder and push your project. You will have to be a collaborator to do this, so if you were not here last time, fill out the attached form to be added as a collaborator:

https://forms.office.com/Pages/ResponsePage.aspx?id=VGZw_NSNrUO32IhD8WGXXigPwIXu8JZHuKz_ckepTQJUNVJNQko2TjJHSlBNSVJZNUxJSDFOVklENS4u

GitHub Link:

https://github.com/tommyrohmann/Coding-Club

That’s all!

So, we made a game. What did we learn by making it?

- How a game is structured in a game loop
- How to use variables and lists
- How to use custom blocks
- Using sensing and operator blocks
- Various loops and if statements
- Signals

As you can see, in coding, a lot of making a project is a bunch of smaller projects being added to a larger whole. While this game is relatively bare bones, we only worked on it for roughly an hour.

With this structure of code, it may seem like adding functionality such as additional states to the game loop can get a little complicated. A big part of coding is learning how to structure the code so it can be easily iterated on and added to.

Next class we will build on what we learned by discussing how we can structure our game loop in a more organized manner, best practices for making our game work, and how we can go about programming various types of game in Scratch.

Thanks for coming!