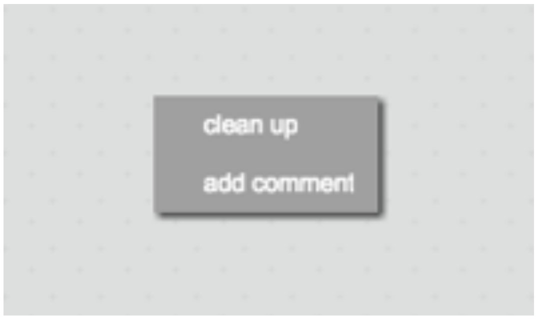
Game On Guide by chaguy2457

Pre-Competition

Step 1: Know the rubric

This is how you're going to be judged! Don't do what I did at Yale last year and finish near last because you completely neglected one of the sections! Here are some parts of the rubric that aren't as intuitive as others:

1. Sprite Orientation (4/100 points): This is quite ambiguous and judges interpret it differently, but I always assume to mean that you should have some rotation of your sprite involved. For example, in Flappy Bird, the bird tilts upward as it flies upward. This applies to both user-controlled and autonomous sprites!
2. Comments on Code (4/100 points): In the coding area, you can right-click to have the ability to add comments to your code (see image). You don't need an extremely thorough explanation. Just write a few words on what part of the sprite this code does (*e.g.*, "Stops the game." "Controls sprite motion."). Not only is this 4% of your grade, but it's really important in the real world of Computer Science, since whenever you share code you don't want your colleagues to spend hours figuring out which line does what. It's annoying but it's the easiest 4 points on the rubric if you ask me.

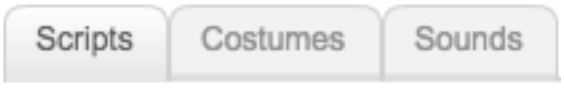


3. Code Organization (6/100 points): Right-clicking in the coding area also gives a "clean up" option (see image). Just click it. Make sure that your code blocks aren't covering the comments! Also make sure the order in which your code is presented is the same order as it is run in the game. Finally name your sprites! And right there is 6 points up for grabs.

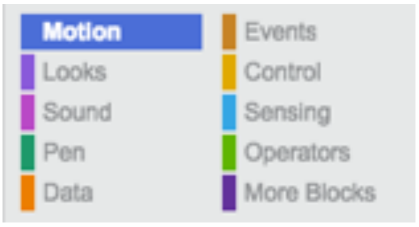
Whether or not you get a rubric during the competition varies from tournament to tournament, so don't count on seeing the rubric while you're coding.

Step 2: Get oriented with Scratch

Scratch has three tabs to work with (see image): Scripts (for code), Costumes (for looks), and Sounds (for sounds! who knew!)



a) Scripts are divided into 10 sections (see image), each of which will be described below:
   i) Motion is a section you'll spend a lot of your time in.



It gives you the option to set the position of a sprite, move it around, or change its orientation. I won't go into what each function does. If you're confused, just right-click and press help to open up the Scratch explanation of the function (which would be better than whatever I could write).

ii) Looks is useful for your title screen and help screen sprites. This is the section where you can control when a sprite appears, how big it is, or, if you're artistically inclined, what color scheme the sprite goes through. You can also have your sprite speak and move it on top or behind other sprites.

iii) Sound is obviously the sound control menu. You can adjust when a sound is played, how loud it is, and how fast it is played. This tab should be used since having sound is part of the rubric!

iv) Pen is where you can have a character leave marks on the background. There are two options: stamp and pen. Stamping is when the sprite leaves an image of itself on the background. Dropping a pen tracks the sprite's movement.

v) Data is crucial for scoring. For those who do programming, this is where you set all of your variables. You can choose to show some, hide some, combine some into a list, etc. More to come on how this section can be utilized later.

vi) Events contains the codes you would put at the top of blocks. It begins whatever code is beneath them when a certain event, such as the green flag being pressed, the sprite being clicked, etc. You can also send "messages" in this category, and basically what these do is that they help you control when a block of code for another sprite is executed.

vii) Control is where you can tell the code to stop for a bit (or forever), loop through a code, run conditionals (if…then, wait…until, repeat…until), and create clones. It's almost like Events in that it controls when code is run but it works more with mathematical conditions rather than universal events.

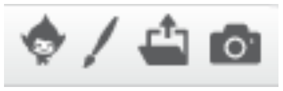
viii) Sensing has a set of useful quantities such as mouse position, position of sprite, and a timer. You can also ask questions and store answers, sense when a sprite is touching a color or another sprite, or when a certain key is pressed.

ix) Operators is all your math stuff. You also have conjunctions ("and," "or," "not") and the ability to concoct strings.

x) More Blocks is for functions. If you find yourself putting down the same block of code over and over, make it a function!

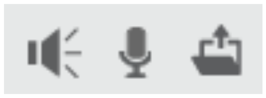
xi) Building code in Scratch is a lot like building Legos. The ten sections are just sections of blocks, and you piece blocks together (drag and drop) to form a coherent block of text.

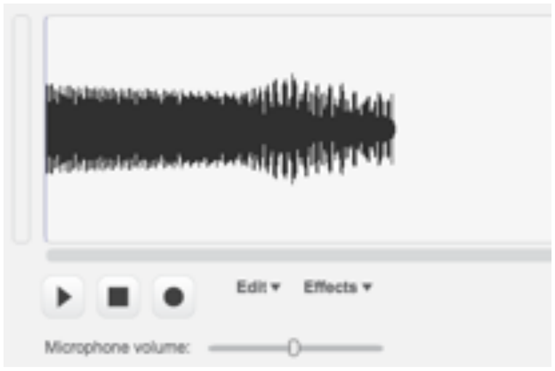
ii) Costumes is where you can design your own sprite, costumes, and backdrop. A sprite and backdrop can have multiple "costumes," which can be toggled via code in Looks. You have four options when creating a new sprite (see image, left to right). You can use a sprite made by Scratch, draw your own, use a photo on your computer, or use your camera. If you use a Scratch default sprite, you only have the option of drawing lines (of varying color and width) on your sprite's costumes. If you decide to make or use your own, you get a cool set of tools, which is listed on the right. If you choose a default sprite or backdrop and add



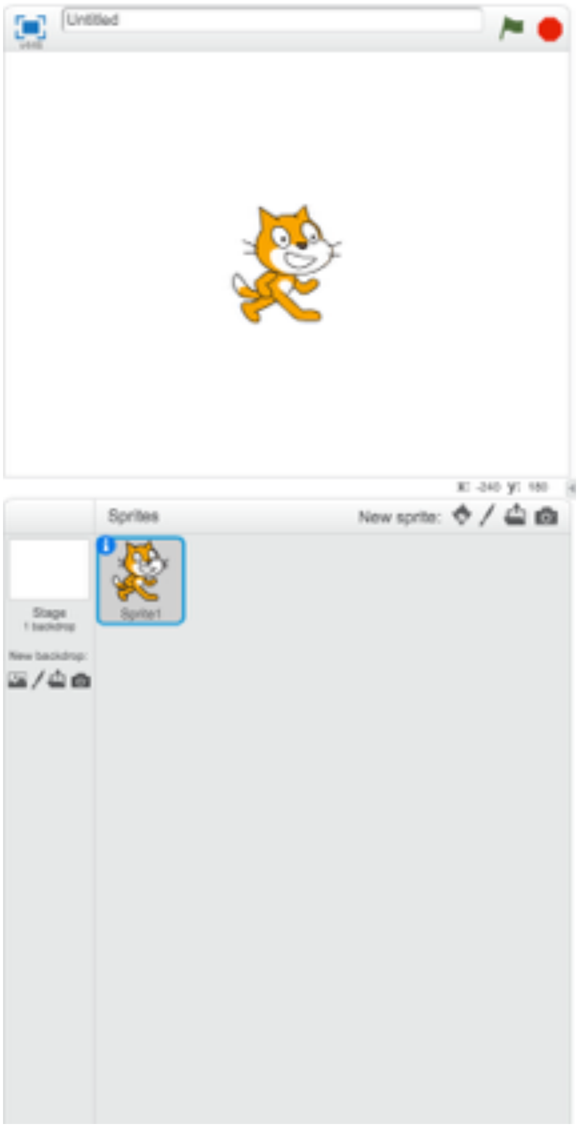
costumes to it you can edit those costumes with these tools as well. Going from top to bottom on the tools list, you have a brush, line, rectangle, circle, text, bucket, eraser, select, background-removing, and duplicating tool. If you're artistically inclined, have at it!

iii) Sounds is where you can customize sounds to be played in your game. The speaker icon (see image) allows you to use and edit a bunch of default sounds, the microphone allows you to record your own, and the folder allows you to upload your own from your computer. In the audio portion of the tab (see image) you can edit and insert effects by clicking and dragging across the portion of the recording you want to edit.





Finally, here is a quick rundown of the other components of Scratch, as shown on the right. The green flag is like a start button; in Events you have the option to run a block of code when that button is pressed. The stop sign stops all codes cold. Below the game screen you have a list of the sprites.



Step 3: Make a game without time constraints

For this go-around, take some time to get acquainted with Scratch and coming up with a game that fits the theme. The theme can be chosen by you or a friend. Themes tend to be generic topics like Election, Dinosaurs, Gravity, Environment, Ocean, etc. I don't really like to write directions on creativity, so all I can do is share my strategy: make a spin-off of a popular game to incorporate the theme, but disguise it enough so that it doesn't seem like a pure rip-off. Just remember that whatever you make, you need a clear objective and a scoring system, preferably numerical. Here are some quick tips on how to make essential components of your game:

1) Title screen: Make sure you have code using blocks from Looks that allows this to appear and disappear. There are many ways to go from this screen to the game or the help screen (using Sensing), but you must have distinct ways for both. I make my title using the text tool in the sprite

maker. I like to make buttons with the circle tool for the transition, but you could also go with a keyboard press and use the text tool to tell the user that.

2) Help screen: Same as above, use Looks to make this appear and disappear as needed. For this screen, just have a button/key press that puts the player right into the game.
3) User-controlled sprite: Unless you want this sprite in the title screen, make sure it's appearing and disappearing regularly. Moving this sprite is probably the hardest part. Motion varies depending on what game you're making, but following the mouse or using arrow keys are all available. Just a tip with having a sprite follow the mouse: the "Go to mouse pointer" block places the sprite right at the cursor, while if you have a block that tells the sprite to "glide" to the mouse's position, you can get a lag in the sprite's movement which makes it seem more natural and enhances the game's difficulty. To get the sprite moving, you can either send a "message" to start movement or set a variable that toggles movement (more reliable in my opinion).
4) Autonomous sprite: These sprites are easier considering their movement is regular. Some people like to have a forever loop or a "wait…until" block so the sprite can keep moving until the game is over. Clones could be helpful too, but make sure the lag involved in creating a clone doesn't mess up the game!
5) Scoring: Remember to reset the score at the game's start and hide/show when necessary.
6) Game Over Screen: Similar to the title and help screens, make sure this shows when appropriate by using "messages" or a toggle variable. If you have various outcomes to a game, I would recommend using costumes so you can quickly toggle to the correct game over screen via messages/variables. The trickiest part about this in my opinion is to have a way for the user to start over. The "cop-out" method is to have a "Stop all" block (which activates the red stop button) and tell the user to click the green flag to start over. The only issue is that it leads the user to the title screen, and I'm not sure exactly how pleased a judge would be with that (I haven't seen many scoresheets). The trickier method that gets the user straight back in the game is through using the messages and variables to "reset" the game to the start of the game.

A big aspect of this is learning how to troubleshoot. Here are some troubleshooting tips.

1) Identify the problem and what part of the code must be interfering.
2) Try figuring out exactly how the code is interfering. You could be missing a block, have an extra block, using the wrong block, ordering it incorrectly, or simply, a typo.
3) To do this you must think like a computer, which is extremely literally. Go through each line and make sure each line follows logically from the previous. This way, you might be able to spot that you are missing something or there are some extras. You might also be able to identify a misused block.
4) An important thing to make sure is that the computer gets the blocks in the right order and when you want them. For example, a "forever" block literally loops that block of code it contains forever until something outside stops it. And you can't do that without stopping the entire code. So if you have code after the "forever" block the computer will simply never get to it. Make sure the order is the way you want it.

5) Finally, and this is the most aggravating, make sure you put in all your numbers correctly, particularly for coordinates. You're not pressed for time yet, but when you are, you might consider doing this first, especially if you know you are prone to typing errors (like me).

Step 4: Make a game **with** time constraints

I would recommend doing step 3 a few times if you're new to Scratch. Once you're comfortable making a game it's time to learn how to make it within 40 minutes (5 minutes planning, 5 minutes testing). Here are some tips on how to get a decent game done in that time span:

1) There are two main game modes: a user avoiding a villain at all costs and a user catching things that are falling down the screen. Choose which one fits the game mode best and quickly start contorting that idea to fit the game mode.
2) You can drag and drop a block of code from one sprite to the other sprite to copy-and-paste. This is especially helpful for creating title and help screens as well as if you are making multiple autonomous sprites.
3) Avoid getting too complicated. The rubric doesn't have anything regarding the number of sprites you have. You only need two: a user-controlled and an autonomous.
4) Start remembering where certain blocks of code are to find them easier.
5) Get used to using a laptop trackpad, since that is the mouse you most likely will be using at a competition (not necessarily invitationals).
6) For troubleshooting, get into a habit you know is efficient and habitual, so problems and the type of problem can be at least identified as soon as possible.

Step 5: Final Preps

Have your strategy set. That includes templates for the title, help, and game over screens, methods for coming up with games and troubleshooting. Once that is done, you can head into the competition confident!

During the Competition

Step 1: Make a title screen and help page ASAP

Have one partner figure out the game while the other already starts making the title screen and help page! It's 10% of your grade! You should have your templates down as mentioned earlier, so this should be done right off the bat.

Step 2: Comment and organize as you go

As mentioned earlier, another 10% of your grade is code comments and organization. Don't wait until the very end to do this; you might not have enough time. Definitely comment as you finish a block and organize each block as you finish a sprite.

Step 3: Do many test runs!

Make sure *every aspect of your game* works. I'm talking about getting started, viewing the title and help screens, movement, scoring, winning, losing, and starting over. If the judge finds any error in your game, you're in trouble, obviously. So avoid that early.

Step 4: If something goes wrong, go back to your troubleshooting skills

If you've finished your game and discover a bug, ***don't panic***. Go back to your methods of troubleshooting and work it out with your partner.

Step 5: If you finish early, get creative!

The rubric for 2015-2016 had a lot of ambiguous adjectives such as "complex" and "creative," whose definition is obviously left entirely to the discretion of the judge. With somewhat creative movement and ideas (which you might end up making under pressure) you're making a toss-up for the judge's favor. So if you find yourself with time left over after you've run a few trial tests, beef it up! Add a second opponent, give the player more options. Compare it to games you have played, and see what this lacks (besides graphics) that stops it from being a top-tier game on the Internet.

Then, I must say, you should probably have a winner, pending the judge's subjective thoughts. If you don't, blame the system; this is by far the most subjective SciOly event anyways. Have fun!