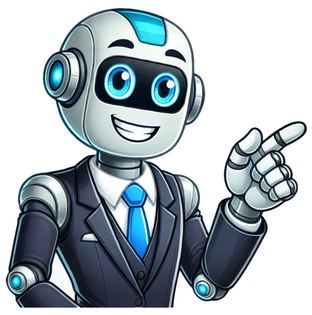
I'm not a bot



You all know about scratchblocks, it is a way to generate images of scratch scripts from text. Currently scratchblocks is being used on the forum and the wiki. It does have some Snap! support, but not a whole lot. There are grey rings, variadic input arrows, and the list icon (although the forum hasn't gotten that update yet). [scratchblocks] (map ((join @addInput :: operators) ((#1) :: grey) @delInput @addInput :: grey ring) over @list :: list) [/scratchblocks] However there are still many issues with the current scratchblocks. Empty rings do not work, the snap "define" block cannot be made because scratchblocks has special formatting for the scratch define block (and there's no way to override it), most Snap! icons are missing, multiline blocks are impossible to make, and more. Seeing as how there is still very little Snap! support in scratchblocks, I decided to create a fork of scratchblocks, calling it snapblocks. Snapblocks aims to make scratchblocks for Snap! as well as fixing many issues. Here are the highlights. If you want to mess around with snapblocks, here's the link to the homepage, although every snapblocks image will be a link to the block on the snapblocks homepage. snap-blocks.github.io Use snapblocks to write pictures of Snap! scripts in forum posts. First and foremost, I added a new Snap! style, which actually looks like Snap! (Scratch 2.0 looks similar, but it's still very clear that it's not Snap! especially when it comes to hat blocks). The scratch define block now has more strict formatting define {block} This will make sure we can create blocks with the define label, with no Scratch define block formatting (yes, I know it will break existing scratchblocks text, but since snapblocks is for Snap! and is seperate from scratchblocks, I feel like this is fine. Plus, I'm not sure how many people ever used it on the forum). define ((block)) [] (() @addInput :: ring) As you can see, upvars now have automatic formatting. The ring around the variable is automatically set to the category of the parent. motion ((var 1)) :: motion looks ((var 2)) ::: looks pen ((var 3)) :: pen stack control ((var 4)) :: control variables ((var 5)) :: variables list ((var 6)) :: list other ((var 7)) :: other Now, on the topic of custom blocks and upvars, I have also added Snap! custom block support. {custom block ((input)) ((number #)) ((bool ?)) :: motion} :: define custom block [text] (5) But Snap! usually shows a + between each block fragment in the block prototype, unless the setting (plain prototype labels) is turned on. Snapblocks does allow the + to be ignored in the block prototype. {+ say + costume + ((costume = costume 1)) + :: looks} :: define say costume [costume 1 V] Unfortunately I was unable to make it automatically set the shape (without making it look bad anyway). (+ sound + ((sound # #)) + :: sound) :: define report (item (sound #) of (my [sounds V])) sound (5) (sound (5)) Now, if you've seen, there are many blocks in Snap! libraries that have multiple lines. Scratchblocks does not have the ability to make multiline blocks, but snapblocks does. (zip (() @addInput) inputs: [] leaf-rank (0) [] leaf-rank (0) :: control) If you put a newline inside a reporter or predicate (oh, by the way, you can now specify the predicate shape, not just boolean), it will have a newline. This is not the only way, as this does not work in command (stack) blocks. Due to this, you can also use , as well as putting a backslash at the end of a line right before a newline. first line second line \ third line \ fourth line On the topic of multiline blocks, I think it's a good idea to mention multiline inputs. (multiline [first line second line third line] :: operators) Snap! also has the ability to put c-shapes in reporters / predicates, which scratchblocks does not have support for, so I added it in snapblocks. Unfortunately I have not added a way to add inputs on mew lines in the middle of a block, specifically there's no way to accurately recreate variadic ring inputs. The only reason I have not added this, is because there's not a really good way of determining if the input should go on the next line at the beginning of the block, or right under the previous input on the same line. If you don't know what I'm talking about, look at the pipe block. I have also added a bunch of Snap! icons (started with the most commonly used icons). @greenFlag @stopSign @turnRight @turnLeft @list @pointRight @turtle @turtleOutline @pause @cloud @cloudOutline @flash @arrowRight @arrowRightOutline @arrowRightThin @delInput @verticalEllipsis @addInput◂⋮▸ :: grey note: there are arrow icons in every direction. This is not all the icons in snapblocks, but all the new Snap! icons have the same name as in Snap!, so if you want to use an icon in snapblocks from snap, just try the name used in snap. Given that you can change the color and size of icons in Snap! I have also added a way to do that. It's pretty much the same way you do it in Snap!. @greenFlag-scale-r-g-b @greenFlag-2-0-0-255 You can also change the scale and color of text, using the exact same syntax that you would in Snap!. $text-2-0-255-0 I have also revamped the block detection system to work with variadic inputs, and blocks that can change their text. if { } @addInput if { } else if { } else if { } else if { } @delInput @addInput all @verticalEllipsis @addInput all (var) and and and @delInput @verticalEllipsis @addInput // Yes, all of these are detected as the same block broadcast [message V] @addInput and wait broadcast [message V] to [all V] @delInput @addInput and wait broadcast [message V] to [sprite V] with data [data] @delInput and wait This new system may be slower than the old method, but I think it's worth it because it allows for easier block creation, and it doesn't actually have to be real time, since most of the time, the image will not change. Block detection explanation I just kind of want to explain how this new block detection system works, as I'm kind of proud of coming up with the idea. Let's start by explaining how scratchblocks detects blocks. We start out with the block information { id: "id", spec: "broadcast %1", inputs: ["%m.messages"], shape: "stack", category: "control", } Note: this is very much simplified, and doesn't contain all the information scratchblocks uses. When you type in a broadcast block broadcast [message v] It creates a block hash from this. A block hash replaces inputs with _ (weirdly similar to how programmatically naming blocks works in snap...) broadcast _ Then it converts the block spec from the block information into this format broadcast %1 --> broadcast _ Then it checks if the two hashes are the same. In this case, they are, so it knows to set the category and shape. This works for scratch, but when you want to add snap primitives into the mix, things get a little messy. For some blocks, I can create aliases, other specs that a block can be compared to. { spec: "broadcast %1", aliases: ["broadcast %1 @addInput", "broadcast %1 to %2 @delInput @addInput", "broadcast %1 to %2 with data %3 @delInput",], ... } Which will then match these using the same rules, just also factoring in the aliases broadcast [message v] broadcast [message v] @addInput broadcast [message v] to [all v] @delInput @addInput broadcast [message v] to [all v] with data [data] @delInput This works for some blocks, but when it comes to variadic inputs that have an infinite amount of inputs, you can see why this wouldn't work. That is why I came up with a more dynamic solution. This method has a base spec, then some spec defs, which replace specific parts in a spec. { spec: "broadcast %1 {receiver}", specDefs: { "receiver": ["", "@addInput", "to %2 {data}",], "data": ["@delInput @addInput", "with data %3 @delInput",], } } This works in a very different way from the old method. Instead of checking if the block specs are equal, it instead creates the block spec based off of what you typed, by stepping through the block spec one word at a time. It starts off with the first word broadcast This is compared with the first word of the spec broadcast (_ to _ @delInput) broadcast (_ {reciever}) As these both have the same thing in the current position, this block is kept, if they're not equal, the block will be dropped from the list of blocks to check. It repeats this process until it comes to a word that is surrounded by curly braces {}, this is a spec def. It then fills this position with it's spec defs. broadcast _ {reciever} broadcast _ broadcast _ @addInput broadcast _ to _ {data} There is only one that can compare to the current block, so the rest are dropped broadcast _ to _ (@delInput @addInput) broadcast _ to _ ({data}) Next it checks for the {data} def broadcast _ to _ {data} broadcast _ to _ @delInput @addInput broadcast _ to _ with data _ @delInput Again, only one of these is the same as the current block spec broadcast _ to _ @delInput @addInput In fact, since they're equal, I don't need to continue, I can just keep this block spec, because I know it's correct. This example didn't have an infinite variadic input, so here's an example. { spec: "{if}", specDefs: { "if": ["if %1 %2 {else}",], "else": ["@addInput", "@delInput @addInput", "else {if}",], } } Let's compare it to this if { } else if { } else if { } @delInput @addInput Which will turn into this hash if _ _ else if _ _ else if _ _ @delInput @addInput The if spec starts with a def, so let's fill that in {if} if _ _ {else} We know it is equal to the first 3 inputs, so let's keep it. Next we need to fill in the {else} def. if _ _ {else} if _ _ @addInput if _ _ @delInput @addInput if _ _ else {if} The last one is the only one that matches out current hash if _ _ else (if _ _ else if _ _ @delInput @addInput) if _ _ else {if} After we fill in that def, we get this if _ _ else if _ _ {else} And then if we keep repeating this process, a couple more times, we end up with if _ _ else if _ _ else if _ _ ({else}) if _ _ else if _ _ else if _ _ (@delInput @addInput) Let's fill in the last {else} and see what we get if _ _ else if _ _ else if _ _ {else} if _ _ else if _ _ else if _ _ @addInput if _ _ else if _ _ else if _ _ @delInput @addInput if _ _ else if _ _ else if _ _ else {if} only the third one matches, so let's use that if _ _ else if _ _ else if _ _ @delInput @addInput And since that one is equal, we know our block is indeed an if block. Again, if at any point, there is something that doesn't match, it will not continue with that block, and will also not be matched. For example, this if { } else { } Will not be matched, because these are not equal if _ _ else _ if _ _ else {if} if _ _ else if _ _ {else} If you want to understand how it's able to match all as well as and and @delInput @verticalEllipsis @addInput, here's the "and" block { spec: "{all}", specDefs: { "all": ["all @addInput", "all @verticalEllipsis @addInput", "all %1", "{and}"], "and": ["%1 and {and}", "%2 @delInput @addInput", "%2 @delInput @verticalEllipsis @addInput",], }, } There are still a bit more that snapblocks does, but this is the basics. As you can tell, I am very proud of coming up with this solution. In fact, I actually came up with it one night, then spent the whole next day implementing it, because I knew it would work. I do really want to make plugins for mediawiki and discourse, specifically fore easier support. Currently, all you really need to do is swap out the scratchblocks files (including the translations-all.js (if you use it)) with the snapblocks files (you may also want to change the reference to use the snapblocks name, everything else can stay using scratchblocks). Since snapblocks is open source, you can contribute to it if you want. github.com Make pictures of Scratch blocks from text. "" I also noticed this when creating my post, `snapblocks` for Discourse I guess I'm that person who finally created a fork of scratchblocks, and made it for snap. Here are some things that I want to add in the future Zebra coloring Block wrapping Multiline comments Maybe a snapblocks to snap xml? (fix the translations system) It would also be great if someone could make a logo for it, as I'm not very good at art. Page 2 bh: smart script pics whats a smart script pic bh: also encourage people to use smart script pics instead I feel like @ego-lay_atman-bay can find a way to store metadata in exported PNG files. But an issue that arises if they did that is the switching input types (switching the set pen color input to a ring, then if you export and re-add it, but it does work somehow) and the @addInput (and other) symbols. It's a script pic that imports the code to Snap! when it's drag and dropped to the window. Yeah, that's what a smart script pic is. In detail, it's a script image, but with XML data embedded into it. You can do this with normal images, but I wonder if will ever have a way to actually read the data. Something like would fit. I don't know about you, but when I'm typing up a block prototype, I see the plusses as something I should type {+ block + prototype + :: motion} :: define rather than {block prototype :: motion} :: define@+ Plus, I feel like more people are going to type the plusses anyway, even if I went with your suggestions. It also adds another layer of complexity, more than snapblocks already adds. The text is supposed to be a direct text representation of the block, which also means that if you type in something, and then snapblocks adds a bunch of stuff you didn't type, it may be a little strange. I am actually not apposed to the idea of adding something to automatically put in plusses between each segment, as that can speed things up, but I do want to allow the user to also be able to manually type in the plusses. tethrarxitet: I feel like @ego-lay_atman-bay can find a way to store metadata in exported PNG files. I actually have plans to generate snap xml from snapblocks, and I do have some ideas on how it could be done. ego-lay_atman-bay: Maybe a snapblocks to snap xml? ego-lay_atman-bay: I don't know about you, but when I'm typing up a block prototype, I see the plusses as something I should type I think you're not considering that + can be (part of) the name of a block, and that such a meaning-carrying + is very different from the syntactic + for places to add inputs, which is why the former are displayed white and the latter black. When I see your example block (not the snapblocks version), I read it as "block prototype," not "plus block plus prototype plus." Especially with dark category colors, the white text jumps out at you, and the black plusses are shyer. This was all carefully designed; Jens and I spent an entire year arguing out the details of the user interface (of everything, not just the block editor) before BYOB 3.0 was released. Maybe you'd prefer the define@plain notation over define@+. That way the default will look like what you're accustomed to seeing. By the way, the reason we have the plain option altogether is that we clearly want the + version for users' block editor windows, but those of us writing documentation or curriculum want script pics of the plain version, so we can focus the reader's attention on what the block actually looks like when used, rather than on the process of creating the block. If we had black plus signs in the blocks in the palette, then I'd see them more as part of the text of the block -- but we wouldn't have black plus signs in the palette! That would be hideous, and would make it harder for a user to read the text of a block. So we want script pics that look like what's in the palette. That's also why I lean toward making plain the default and requiring @+ for the plus version in Snapblocks: When people put a script in a forum post, they want it to look like the script in the scripting area, most of the time, not like a block editor window. But, I dunno, maybe I'm wrong about how users think. Anyone else have an opinion? When I see plain prototype labels, I think Hmm. Why are there no plusses? Weird. My vote (that is, if I have one) goes to define@plain. {+ say + costume + ((costume = costume 1)) + :: looks} :: define say costume [costume 1 v] bh: I think you're not considering that + can be (part of) the name of a block, and that such a meaning-carrying + is very different from the syntactic + for places to add inputs, which is why the former are displayed white and the latter black. I actually did think about that, I just never said anything. My idea would be for users to use /+ to force that to be part of the block. Here's ny full idea of what I'm leaning towards. It ignores plus signs that are between two block fragments. If there are 2 plusses in a row (with a space between), the second one is part of the block. It pretty much alternates for more plusses in a row. If there is \+, it forces that plus to be part of the block. If there is @+, it's the special formatted plus that does not count for the block. If the user does not want to type in any of the plus signs, yet they still want them, they can add the override define@+, or maybe just define+. I feel like that's the best of both worlds. There's the option for users who didn't read much of the documentation, and an option for users who read all the documentation. bh: When people put a script in a forum post, they want it to look like the script in the scripting area, most of the time, not like a block editor window. I don't know about anyone else, but I want my block prototypes to look like the block editor, not the palette. I also want my scratchblocks text to look like the block that I'm creating. I feel like that might also be easier to understand for some users, than some special override. ego-lay_atman-bay: I want my block prototypes to look like the block editor, not the palette. Same. Huh. Okay, I'm outvoted. I guess I'm getting rusty at thinking like a user. :~) For @ego-lay_atman-bay: (the script @addInput { move (10) steps } :: reporter operators) (the script. Input names: ((a)) @delInput @addInput { move (a) steps } :: reporter operators) (the (this script :: control) block @addInput :: ring operators) (the (b) block. Input names: ((b)) @delInput @addInput :: ring operators) Can you implement a new ring rounding from Super-Snap!? Only for "the script %inputNames %c" ring is the normal rounding divided by 3. Potentially, as I'm currently redoing the snap rendering to use pretty much the exact same drawing functions as snap (just translated to svg). Oh, and I'm fixing the multiline block lines to have consistent spacing. Alright, I was able to add it. This is also kind of a sneak peak for my new redone snap style rendering, which you may notice the upvars look much better (and there's less padding in general). That's so good, I'm giving to you back the developer mode! Edit: After all, now the developer mode in Super-Snap! is available. To be honest, I literally used almost the exact same functions that snap uses to draw the reporters, which means, I was able to just change the rounding value to what super-snap uses. Super-Snap! is obviously a Snap! mod that gives you more features and old BYOB ones. That means that old BYOB users can migrate their projects to that Snap! mod. Alright, I finally pushed my changes to the snap style. This really does look a whole lot more like snap. Guess which one is snapblocks (without looking at the filename). I also used pretty much the exact same functions snap uses to draw the blocks, which really helped make sure reporters and predicates look right. Just saying, I still want to move these changes to the scratch 2 and 3 styles before a new release, plus I want to add the custom block prototype + formatting. Page 3 I've been thinking about a way to allow readonly number dropdowns, and editable text dropdowns. My idea is to use a capital V for readonly, and a lowercase v for editable. It would look something like this. switch to costume [costume V] (split [hello world] by [v]) (item (1 v) of @list) (readonly number (1 V)) The filetype name defaults to some hex gibberish. :P I was referring to how sometimes the forum shows the filename in the posts, but I guess it didn't here (it's very inconsistent when it allows you to click on the image, and see the original filename). I assume it depends on the size of the image. You can select the image and quote it to see the image name, though. Interesting. What about if the image can be imported to Snap! then you're able to get the blocks of it? (while kinda complicated but still kinda cool) ego-lay_atman-bay: I actually have plans to generate snap xml from snapblocks, and I do have some ideas on how it could be done. ego-lay_atman-bay: Maybe a snapblocks to snap xml? I think it might be only draggable when you drag and drop it in, and clickable when you use the upload button. Drag&Drop: Upload Button: I admit defeat I don't know how it works I finally added block wrapping support (I stole this script from Pocket size hyperoperator script) Yeah, there are a few things that need to be fixed, but at least it works. For devs Block wrapping does need to be enabled as an option, where you specify the style var docView = snapblocks.newView(doc, { style: "snap", scale: 1, wrap: true }); The wrap size is also customizable var docView = snapblocks.newView(doc, { style: "snap", scale: 1, wrap: true, wrapSize: 100 }); Oh yeah, I also added proper scratch 3 rings, mainly just to have them look better than what they were before. I finally added zebra coloring! The only issue is that the colors aren't exact, but that's due to the fact that the svg color mix function is different from snap's color mix function. If I put the category colors in the code instead of in css, then maybe I could put snap's color mix function in. I guess that's another project I could work on. For devs This also has to be enabled var docView = snapblocks.newView(doc, { style: "snap", scale: 1, zebraColoring: true }); ego-lay_atman-bay: I finally added zebra coloring! Yay!! I've all along been wondering why anyone uses Snapblocks given that we have smart script pics, but zebra coloring is a tipping point; it's really starting to look like Snap!. Maybe it could even include the XML for the smart script pics sometime, that would be Tipping Point: 2 insert_user_name: Maybe it could even include the XML for the smart script pics sometime, that would be Tipping Point: 2 Yes, although real smart script pics have the advantage that they include in the XML all the dependencies of the visible script. One made in Snapblocks wouldn't be runnable. bh: Yes, although real smart script pics have the advantage that they include in the XML all the dependencies of the visible script. One made in Snapblocks wouldn't be runnable. That is very true. Fortunately I can change the app name in the xml to make snap show a popup warning that the script was created in an external program, and may not be runnable. As for embedding custom blocks, the only custom blocks that would be embedded would have to be defined in the same snapblocks script. There is an old project based on Tim Radvan pull request@2017 to export scratchblocks from Snap IDE. BTW, Snapblocks are really obsolete for a Snap. They need dedicated adapters for every platform/transport (Discourse, Discord, HTML, email, text editor). But it's your circus, these are your monkeys... dardoro: Snapblocks are really obsolete for a Snap. They need dedicated adapters for every platform/transport (Discourse, Discord, HTML, email, text editor). True, but I'd rather have something similar to scratchblocks, but for snap, as there was pretty much nothing before. I actually fully support smart script pics way more, and I prefer to use those over snapblocks. As for the things in the list you mentioned, uh, HTML is what snapblocks is made with (it uses javascript to generate SVGs, aka, HTML), there is a discourse plugin being worked on, and I don't see much use for discord or email (scratchblocks doesn't even have a discord bot, so I'm not sure it would be useful). Text editor may be useful, but again, there's nothing like that for scratchblocks. I know you didn't mention this, but there is actually a mediawiki plugin that is already made (and I hope it works), it's a fork of the scratchblocks mediawiki plugin. Alright, I finally got around to adding the plus sign formatting (and fixed control custom block prototype zebra coloring). {test block} :: define+ {+ test + block +} :: define {test block} :: define {\+ test \+ block \+} :: define {+ + + test + + + block + + +} :: define {@+ test @+ block @+} :: define Oh by the way, I actually changed how menus are made. A lowecase v is an editable menu, whereas a capital V is a readonly menu. [edit v] [readonly V] (2 v) (5 V) ego-lay_atman-bay: Alright, I finally got around to adding the plus sign formatting (and fixed control custom block prototype zebra coloring). Cool! Does @+ mean the same as +? bh: Cool! Does @+ mean the same as +? It means that it's the dark + no matter what. {@+ @+ test @+ @+} :: define {\+ \+ test \+ \+} :: define {+ + test + +} :: define Page 4 ego-lay_atman-bay: It means that it's the dark + no matter what. Ah! I see. Thanks. Alright, I've added block wrapping and zebra coloring to all the styles, so now I'm going to try to get ready for a 1.1.0 release. I have released v1.1.0 with the changes that I've been posting about. GitHub Alright this is a big release v1.1.0 Additions Added new snap-flat style. This is the Snap! flat design style. Added zebra coloring. This needs to be enabled with the zebraColoring option. Added b... Additions Added new snap-flat style. This is the Snap! flat design style. Added zebra coloring. This needs to be enabled with the zebraColoriing option. Added block wrapping. This needs to be enabled with the wrap option. You can also set the minimum width needed to wrap with the wrapSize option. If you don't know how to use these, you can look in the index.html file in the main repo (hopefully I can create some actual api documentation to make it easier to find). Added special Snavanced (formerly Super-Snap!) this script block support. Added + formatting in custom block prototypes. If you set the override to define+ it will automatically place all the plus signs between the fragments. \+ will force the + to not be formatted, likewise @+ will force the + to be formatted (it is technically an icon). Add @camera, @circle or @circleSolid, and @notes (♫) icons. Changes This is very important. I have changed snapblocks to export as snapblocks isntead of scratchblocks. If you want, you can just do var scratchblocks = snapblocks to easily transition to this new update. C-shapes are created if there is a newline after a {, otherwise it's an embedded script (for rings). Any menu can be editable or readonly. Capital V is readonly, and lowercase v is editable (all menus are readonly in the scratch 3 style). Fixes Updated Snap! style to look more like Snap!. Fix grey rings in the Scratch 3 style. Fix lines in multiline blocks to have consistent spacing. Multiline grey rings are now round. Multiline hat blocks now have the hat block top (needed for block variables in custom block prototypes). Fixed the if else reporter not detecting. snap-blocks.github.io Use snapblocks to write pictures of Snap! scripts in forum posts. This is very fun! However, if I export to SVG and put it on the forums they look BLACK. Also, if I use the code: return [] It won't become control-cap, but instead red. ego-lay_atman-bay: Added zebra coloring. This needs to be enabled with the zebraColoriing option. Two "i"s, really? That was a typo, it's zebraColoring I just released v1.1.1 which fixes an error with comments. GitHub v1.1.1 Fixed Error when rendering comments in the snap style. Fixed comment y to actually be in the middle of a block. I found your Snapblocks Extension in Miraheze Yup, @r4356th requested it, and we both worked on the actual extension. Could a documentation/tutorial page be created for this similar to this wiki page? Block Plugin - Scratch Wiki (I could make it if nobody else wants to) I'm actually working on one right now, but instead it's going to be on a new page on the snapblocks site (mainly because the snap wiki hasn't gotten snapblocks yet). Hey guys, I finally got a documentation website set up (or at least the repo, I need to do a bit more work to get the website up). If you want to help me fix up the documentation, please do. github.com Documentation for snapblocks Alright, I had to fix a few things with the actual website before it could be published (serverside rendering was not working). Here it is. Hopefully I can actually get to work on the actual contents of the site now. snap-blocks.github.io Snapblocks is a fork of scratchblocks that adds all the features found in Snap! Text is broken in the docs: (but on the snapblocks itself) Yes, I know why that's happening, and it will be fixed in snapblocks 1.2.0. Just show the first snapblocks result, it fixes it. wtf this is a scatchblocks mod? i thought this was a script pic! Well I guess that's a sign that it's very accurate. Did I do something wrong? This is my code: when flag clicked move (10) steps snapblocks.renderMatching('pre.blocks', { style: 'snap', // Optional, defaults to 'snap'. languages: ['en', 'de'], // Optional, defaults to ['en']. scale: 1, // Optional, defaults to 1 }); and this is the result It's because you're loading the language de without the translations-all.js file. If you remove de, or add the translations-all.js file, it will work (and if I'm being honest, you probably don't need translations, as they are kind of broken). Page 5 i mainly don't like using script pics because it requires me to grab my mouse (i was using the keyboard before) go to snap wait for it to load go through all the tiny little menus and look through projects and such to get real versions of blocks sometimes create fake versions of blocks, constantly switching between keyboard and mouse to create the block definitions build the script switch tab to the forum again because i forget what's in one of the posts switch tab back to snap select a place to save the picture go find the picture file close snap yes, close snap without saving upload it to the forum delete the picture file if script pics could go straight to the clipboard it would help somewhat. I like creating script pics, because it allows me to test the script before sending, to make sure I don't make a mistake. Plus, I find it easier to create scripts while looking at the block pallette, instead of having to remember what every block is named, and what they do. sarpnt: go to snap wait for it to load ... close snap yes, close snap without saving To be honest, I just leave a snap tab open at all times, so it's easy for me to switch between. sarpnt: if script pics could go straight to the clipboard it would help somewhat. Ok, that would actually be a really cool feature, and I would use it over my ever growing script pics folder. Seeing as how @jens has added the ability to copy the text from a report bubble, I have a feeling he would be more inclined to add that. Just to be clear, this post is all about my personal opinion, and how I do things. I'm not saying you should do it the way I do it, or that my way is better, I'm just saying that I find it easier to use script pics. ego-lay_atman-bay: To be honest, I just leave a snap tab open at all times, so it's easy for me to switch between. i have my browser security settings maxed out, to use snap i have to go into some special menu to start a new browser profile and open snap in that, then make sure not to accidentally open other things in that window. i also already have 896 tabs open :) sarpnt: i have my browser security settings maxed out, to use snap i have to go into some special menu to start a new browser profile and open snap in that, then make sure not to accidentally open other things in that window. Oh wow, I guess that is difficult. sarpnt: i also already have 896 tabs open Really? I only ever get that many tabs on my phone, because I always start fresh every time I turn my computer on. sarpnt: i mainly don't like using script pics because it requires me to Ah, I rarely find a need to create a script pic from a project I don't already have open. It's because I'm working on a project, or reviewing someone else's project, that I want to post a picture of a piece of it. Argh! My snapblocks.min.js didn't save and github is firewalled, can you send the code in a [details] thing Sorry, it's too big to send. However, if npmjs.org isn't blocked, you can grab it there. it isnt blocked, whats the link Nvm i found it @ego-lay_atman-bay I can't find a link to download it other than github The other download link can be found here, snapblocks - npm (in the build folder) I already looked and I couldn't find any links there, even in snapblocks.min.js You're supposed to download the snapblocks.min.js file from there, and put it on your website. im at home now so its fine it::sensing freaking::operators works::control Hey also, dunno if there is any official way to make define hats but i did this (+hello+::sensing command)::control hat (+hello+::sensing command)::control hat // how the same code looks in scratchblocks also .io links are blocked so i cant access the guide ON MY CHROMEBOOK** There is an official way to make hat blocks, and it is similar to how you did it. { + hello + :: sensing } :: define hello Of course you also don't need to type in all the + signs manually { hello :: sensing } :: define+ hello mr_owlssssnap2: also .io links are blocked so i cant access the guide ON MY CHROMEBOOK** Don't worry, the guide is very very barebones, and right now is pretty much a direct copy from the scratch wiki article. Although, there is kind of a guide in the first post of this topic (there are some small changes due to later updates, but it's still very relevant). (hello::sensing command)::define+obviously doesnt work with scratchblocks Page 6 i unminified it and can I add changes? What changes? If you're planning on modifying snapblocks, it's best if you just create a fork of the source code, and modify that, as that has actual variable names (the minified version makes every variable name a single letter), and it is easier to find stuff. I also have an idea- you could add result pics by doing this! ((1) + (2)) //$ 3 With the block: (hi!::operators) //$ hello! With the block: I don't feel like adding result pics, as there's more in results then just text, and it would just get super messy. Plus, I prefer using actual result pics, as those will always give the correct result. Snapblocks should not be used to help people with a script, as you're more likely to use snapblocks without testing the script beforehand (that's why script pics are better, they allow you to test if the script actually works). I finally came back, after the wiki got snapblocks working, and I finally added the ability to show spaces. (split [hello world] by [v]) ego-lay_atman-bay: the ability to show spaces. Cool! I keep having this feeling like you're going to get to where you've written 96% of a Snap! evaluator (e.g., users want result pics, you don't want them posting wrong results, so you add a "show result" flag and compute the result yourself...) and then you can just add the last little bit, and we can retire... bh: and we can retire... oh no you dont, we need an admin Oh no, I've reached a conundrum. Snapblocks cannot tell the difference between the snap list contains block and the scratch text contains block. // snap list contains // scratch text contains // snap list contains, or scratch text contains // scratch list contains There's no way to determine whether is the snap list contains, or the scratch text contains, as it can be either block. Snapblocks determines ambiguations like this by checking the input types, or the values of the inputs. For example, the of block. There's 4 of them, and the only way to determine what category it should be, is by looking at the value of the first input (thankfully there's only 1 of block that you can stick reporters into the first slot, so I don't run into that issue here). I'm inclined to make be the snap list contains block, as this is snapblocks, and you're less likely to be writing scratch exclusive blocks in snapblocks. Hmmm... If is "snap list contains", then or could be "scratch text contains". No, cause curly braces represent a stack block. Parentheses represent a round number input or a reporter. I don't want to add more syntax, as no one would use them, because people are way more used to scratchblocks. ego-lay_atman-bay: cause curly braces represent a stack block. Parentheses represent a round number input or a reporter. [ ]? Those are text. I just looked at scratch, and the scratch text contains block has a question mark at the end, so now I realize my issue wasn't a real issue. All I need to check, is if it has a question mark (snapblocks doesn't bother checking for the question mark in many blocks). // snap list contains block // scratch text contains block Woops, need to fix this. Comments don't have whitespace bullets, is that what you mean? I'm just now thinking of making snapblocks not ignore question marks... Im not sure what the problem is, why dont you just remove the scratch blocks version? I'm not removing the scratchblocks version, because it would be too much work, and it would also break existing scratchblocks (more than snapblocks already does). Plus, why not also support scratch? Many blocks in snap are also in scratch, so I might as well just support scratch and snap. Page 7 ego-lay_atman-bay: (snapblocks doesn't bother checking for the question mark in many blocks) Why on earth make this a special case? I mean, why not get the entire title text to match? Especially if you want this to work for custom blocks, which might be written by someone who speaks a verb-at-the-end language. Yeah, especially with this. ego-lay_atman-bay: I'm changing that in 1.3.0, which will make it more strict, but also better. Oh, and by the way, apparently it was also ignoring these characters, ,%?: (which it won't ignore in 1.3.0). I think the reason why scratchblocks was ignoring those characters, was so scratchers could type go to x () y () instead of go to x: () y: (). ego-lay_atman-bay: I think the reason why scratchblocks was ignoring those characters, was so scratchers could type go to x () y () instead of go to x: () y: (). Oh yeah, that makes sense. I was thinking it would find the nearest match. Any recent visual updates? Well, I'm currently redoing the list icon, and I'm also tweaking the input top and bottom padding. I also have an idea for making the zebra coloring more accurate, but I don't know if I want to get into that right now. Oh, and by the way, I also made sure every single snap, scratch 2, scratch 3, and scratch 1.4 block is detectable (I know barely anyone would use blocks from scratch 1.4, I just added them because I can). ego-lay_atman-bay: I know barely anyone would use blocks from scratch 1.4 I use Scratch 1.4/BYOB 3.1, occasionally. I thought it was just Snap! and Scratch 3, but not Scratch 2 OR 1.4!!!!!!!! What's next, Snap! Dev? Scratch Pre-1.4? BYOB? Scratch's Predecessor? Squeak? Smalltalk-80? sonic123cdmania: Squeak? Smalltalk-80? Those are text-based. No need for a rendering program for text :) (Okay, not true. but not this rendering program) sonic123cdmania: was for bluebaritone21: I use Scratch 1.4/BYOB 3.1, occasionally. . sonic123cdmania: I thought it was just Snap! and Scratch 3, but not Scratch 2 OR 1.4!!!!!!!! What's next, Snap! Dev? Scratch Pre-1.4? BYOB? Scratch's Predecessor? Squeak? Smalltalk-80? was for ego-lay_atman-bay: Oh, and by the way, I also made sure every single snap, scratch 2, scratch 3, and scratch 1.4 block is detectable (I know barely anyone would use blocks from scratch 1.4, I just added them because I can). . Scratch 2 is in scratchblocks, and is the style the scratch forums still use (they haven't updated in a looong time). There is no scratch 1.4 style, but there were some blocks from 1.4 in scratchblocks (although they were set to the obsolete category), namely the forever if , and the if (no then. That's why the snap if block kind of worked before). bluebaritone21: I use Scratch 1.4/BYOB 3.1, occasionally. Honestly I didn't think about adding the BYOB blocks, even though I opened BYOB when adding the scratch 1.4 blocks (I opened BYOB so I could get the blocks selector). Although, I'm pretty sure most BYOB blocks are in snap anyway, so there wouldn't be much for me to add. I forgot to mention, I actually added some hidden scratch blocks, like the while block, and scratch's hidden for in block. I added those, because I've used them many times in my scratch projects before (you can find them in turbowarp). ego-lay_atman-bay: every single snap, scratch 2, scratch 3, and scratch 1.4 block is detectable ego-lay_atman-bay: There is no scratch 1.4 style, but there were some blocks from 1.4 in scratchblocks ?????????????????? I can put the scratch 1.4 blocks into snapblocks without the scratch 1.4 style. The blocks are not tied to the style (it can detect snap blocks in the scratch 3 style). Ah, ok, but there's still the question sonic123cdmania: What's next, Snap! Dev? Scratch Pre-1.4? BYOB? Scratch's Predecessor? Squeak? Smalltalk-80? . ego-lay_atman-bay: (you can find them in turbowarp). Really? I got them from editing the project.json, back when I used Scratch. I'm not adding snap dev blocks, as those can change any time. The only snap dev block I added, was renaming the primitive block to extension, as that seems like it's going to stick. I don't know what scratch's predecessor is, and I have absolutely no plans to add squeak or smalltalk. sonic123cdmania: Scratch Pre-1.4? BYOB? Scratch's Predecessor? Squeak? Smalltalk-80? What about these? Sorry, I edited that into my previous post. ego-lay_atman-bay: I don't know what scratch's predecessor is, and I have absolutely no plans to add squeak or smalltalk ego-lay_atman-bay: I have absolutely no plans to add squeak or smalltalk Ok. ego-lay_atman-bay: I don't know what scratch's predecessor is I think it was Logoblocks. Plus, what about Scratch Pre-1.4 and BYOB? Page 8 If someone wants to show pre-1.4 blocks, then they're gonna have to make those themselves (by setting the category and shape). Since not many people use those, there's not really any reason to add those. As for BYOB ego-lay_atman-bay: Honestly I didn't think about adding the BYOB blocks, even though I opened BYOB when adding the scratch 1.4 blocks (I opened BYOB so I could get the blocks selector). Although, I'm pretty sure most BYOB blocks are in snap anyway, so there wouldn't be much for me to add. ego-lay_atman-bay: most Not all. basically the only ones are [scratchblocks] (the () block::operators) the script{::grey cap }::operators hat cap [/scratchblocks]. And, you can make them already I did actually add special corner roundness for multiline ring blocks, specifically for snavanced (formerly super-snap). ego-lay_atman-bay: There is no scratch 1.4 style Snap! style is pretty much that, isn't it? That is a good advance, but Snavanced! "was" formerly Super-Snap!, I deleted everything about Super-Snap! before I am doing the math update. No, there are many different things that don't look like snap, like hat blocks, number inputs, block shading, and many other things that make it not look like snap. Snap is just generally a lot cleaner than scratch 1.4. ego-lay_atman-bay: Snap is just generally a lot cleaner than scratch 1.4. Ah. Of course BYOB was exactly Scratch 1.4 style, and Jens was pretty much trying to duplicate it in Snap!. I do remember that hat blocks took a long time to get just right. I just looked at byob, and found that these are the only blocks I need to add bh: Jens was pretty much trying to duplicate it in Snap*!*. I do remember that hat blocks took a long time to get just right. Here's what hat blocks looked like Compare that to snap Scratch 2 hat blocks look more like scratch 1.4 than snap I'd say snap looks the best. As for other blocks, I feel snap still looks different from scratch 1.4 Oh yeah, by the way, snap actually has a slight drop-shadow on the text, which BYOB (or scratch 1.4) doesn't have. Seriously, that's a tiny detail that I wish scratch would do. I also found: Individual true and false blocks in BYOB3. ego-lay_atman-bay: It would also be great if someone could make a logo for it, as I'm not very good at art. I made a logo for snapblocks: (font is Montserrat) Does it look good for you? If it isn't, can someone redesign the logo? Oh yeah, I redesigned the previous logo and now looks like this: How about bluebaritone21: This one has little or no decoration. What about: I remade it, feel free to edit SVG: PNG: Font is also Montserrat Google Fonts The old posters and signs in the traditional Montserrat neighborhood of Buenos Aires inspired Julieta Ulanovsky to design this typeface and rescue the beauty of mr_owlssssnap2: PNG: Why is this icon aliased? I can see the pixels on it, where they start and stop abruptly without fading. I say make the logo with the program itself, spell out snapblocks out of textboxes or something. The swoosh of the hat block has jaggies, that's what @tethrarxitet is seeing. If you guys are going to mess with fonts, italicize the exclamation point! :~) Page 9 bh: If you guys are going to mess with fonts, italicize the exclamation point! :~) Or maybe use the Verdana font, the font that snap uses. ego-lay_atman-bay: Or maybe use the Verdana font, the font that snap uses. Or Candara, the font in the Snap! logo. How about this? It was generated by snapblocks, and manually modified to make the hat block blank (snapblocks makes empty blocks very skinny, so I had to put some text in the hat block to make it full size). (Please excuse the bad editing. I haven't got around to compiling a real photo editor from source.) Don't worry, I can't even edit snapblocks svg in inkscape anyway, as it does a bunch of stuff that inkscape doesn't like. How did you make the words "Snap!" and "Blocks"? Does whatever typeset those words not do italics? Those just use the svg element, which you can use regular css text styling on. Here it is with making it italic. Keep in mind, I had to separate the ! from "Snap", so I just magic numbered it's x position. edit: and now that I'm on my phone, which doesn't have the ver