Person-v132’s Game On Guide
Revised 07/01/2019

Introduction

Hi! I’m person-v132, a soon-to-be 8th grader at Creekside Middle School. I did Game On for the 2018-19 SciOly season, and here is a compilation of tips and tricks my partner (Phenakism) and I learned along the way. This guide is not a strict set of rules to follow, rather a general guide to get your team started. Feel free to adapt any of these tips to best suit your needs.

Game On is a lab event where competitors have 50 minutes to create a computer game based on a scientific topic using Scratch. Scratch is a free, online graphical programming tool commonly used to teach programming to kids, but can also be used to create very advanced applications. You will not know the scientific topic until the event starts.

If you haven’t already, take some time to get familiar with Scratch. I would recommend following a simple game tutorial, then play around a bit yourself to see what each block does. Scratch is online here: https://scratch.mit.edu

You should also read through the rules and rubric to understand what you need to do, especially the Rubric-Explained here: https://www.soinc.org/game-b

This will also help you understand what I’m talking about. Game On is also one of the most subjective events there is, so many things will be based on the judges’ opinions. When grading your own projects, it is better to grade as strict as you can really help you identify weak points and improve on them. Your judge may be really mean or really nice, so prepare for the worst possible situation by being super critical.
Table of Contents

Introduction 1

Table of Contents 2

The Basics 4

Roles 4
  The Coder 4
  The Designer 4

Scheduling/Time Management 4
  Stage 1: Design and Prep (10-20 min) 5
  Stage 2: Game Creation (20-30 min) 5
  Stage 3: Cleanup/Testing (10-15 min) 5

Setup Guide (Stage 1) 6
  Step 1: Backdrops 6
  Step 2: Buttons 6
  Step 3: Design 8
  Additional Features 9

Game Guide (Stage 2) 9
  Step 1: Create the Sprites 9
  Step 2: UC Sprite 10
  Step 3: Autonomous Sprites 11
  Step 4: End of Game 12
  Additional Features 13

Debugging 13

Grading a Game 15

Preparing for Competition / Tips 16

Misc. Tips and Additional Resources 17
  Phenakism’s Way to Draw Efficiently 17
Things to Keep In Mind

Other Game Types
  Shooting:
  Platformer:

Helpful Links

Conclusion
The Basics

Since you only have 50 minutes to complete a game, efficiency is of the essence. You also only have one computer to be shared across two people, so you and your partner need to cooperate well and manage time effectively.

Roles

To allow for easier planning/practice, each person should have a role. The tasks for each role changed slightly as we honed our skills. Feel free to tweak this as much as needed to suit your needs.

**The Coder**

The Coder is the one responsible for setup, cleanup, and other tasks needed by the designer. The coder should be very familiar with the default setup procedure and should be able to finish it in under 10-15 minutes. The coder should also be very familiar with the rules and have a mental checklist for each item needed within the game. The coder should be able to type quickly with minimal mistakes as well.

**The Designer**

The Designer is responsible for coming up with ideas for the game. The designer should have very broad scientific knowledge in many different fields. They are also responsible for creating the main game portion set up by the coder and should be very familiar with a default game template.

Scheduling/Time Management

Early in the season, you and your partner should work on getting familiar with Scratch and practicing with making basic games. As competitions start to approach, you should start planning a schedule to help you stay on track. This is what worked for us.
• **Stage 1: Design and Prep (10-20 min)**
  During this stage, the coder should **immediately** get to work setting up the backdrops, broadcasts, sounds, buttons, and other game items. I will go into the setup procedure more in-depth later. While the coder is doing this, the designer should be brainstorming ideas for a game that follows the scientific topic and four scientific facts relating to the theme. The designer should also be thinking about how the game will run and the sprites/variables/elements needed. Blank paper will be provided for planning. After the coder has completed the first part of the setup, the designer tells the coder about the theme and the coder fills out the backdrops accordingly.

• **Stage 2: Game Creation (20-30 min)**
  This stage is where the game is created. When the coder hands off the keyboard/mouse to the designer, the designer should start by creating all the necessary sprites. The designer should also have the default game template memorized, as they got rid of game types. I will go more in-depth about the game templates later.

• **Stage 3: Cleanup/Testing (10-15 min)**
  For this final stage, you should **always aim to allocate at least 10 minutes** at the end to make sure your project is ready. While you should be adding comments/testing each part as you go, this is your last chance to make sure everything works and is ready to go.

Using this schedule, each partner spends a somewhat equal amount of time at the computer. While this does introduce a switch which costs some time, we found that switching halfway lets each partner rest, as furiously clicking under pressure is surprisingly tiring. If you or your partner can drag and drop blocks for 50 minutes straight without fatigue, feel free to have one dedicated computer person.
Setup Guide (Stage 1)

https://scratch.mit.edu/projects/318928715/

The project linked above is a sample template for setup. As you can see, it already contains many of the items required on the rubric, including the introduction/help page, sounds, comments, and scientific explanations. The black backdrops, colors, and button hover animation also help build up judge impressions. Here, I’ll walk you through the setup process.

Step 1: Backdrops

The moment the event begins, the coder should create the backdrops. After deleting the Scratch Cat, create five blank backdrops. We usually name them “Start”, “Help”, “Game”, “Win”, and “Lose”, respectively. During this stage, you can also choose some default background music and put it in a “forever” loop in the backdrop’s code.

Step 2: Buttons

After the backdrops are created, create a button sprite. We use the default “Button3” sprites from the Scratch library, as there really isn’t a point incentive to draw your own buttons. The reason we only need one button right now is that once you’ve set up one button, you can duplicate it to save time. After the button has been created, drag and drop the button’s code as follows. If you think you have time, add a quick sound effect for bonus points. This code is for the “Start” button.
This first stack of blocks sets up the button and backdrops for the start of the game. It switches the backdrop to the start screen, tells the button to go to the bottom left-hand corner, and show. The forever block at the end runs an animation that increases the size of the button when hovered over. This isn’t required, but it’s nice to have.

This next stack of blocks starts the game when the start button is clicked. It also switches the backdrop to the game screen.

These stacks of blocks change the button’s position for the help screen and the Game Over screen.

Be sure to add comments as you go.

After coding the Start button, go to the costume editor and add the text “Start” to the button. Then, duplicate the button to create the “Help” button. Rename the new button and make these edits to the code.
Step 3: Design

Now it’s time to make everything look fancy. By now, the designer should have a game idea ready. Go to the backdrop editor and open up the title page. Using the rectangle tool, draw a large rectangle big enough to cover the entire page. You can make this any color you want, but I usually default to black. With the background color done, we can add a creative title and subtitle. The subtitle isn’t required, but it’s fun and may add to the judges’ impression.

For the instructions page, **four things are needed. Game objective, movement controls, scoring, and scientific explanation.** If these things do not fit on the page, a third page can be used for just scientific explanation, but this adds complexity and costs time, though it may be worth it to really make sure you get those points.

You should have **at least four facts** about the scientific theme and how it relates to your game. This clears up confusion for the judges and will help to avoid missing out on those 12 points or getting tiered.

The win/lose screens don’t really need much as long as it’s clear which is which.
Additional Features

Besides just this basic setup, more detail can be added if there is extra time. Sound effects when the button is clicked can also be used for sound variations.

Game Guide (Stage 2)

https://scratch.mit.edu/projects/297508657/

The project linked above is a practice game my partner and I made when preparing for competition. Feel free to take a look around and use this as a template for practice. I also may not have covered all the blocks, so do check it out if you need further help.

Because they got rid of game types for the 2019 season, we did not have to worry about each game types. Instead, we focused on a single game type to practice and memorize. We chose simple collection/avoidance games as the type as it has all the collision points we need. However, other game types such as a shooting or maze may be more unique and stand out. In this guide, I'll talk about collection and avoidance, though if you want to learn about other game types I'll link to some examples at the end.

Step 1: Create the Sprites

Once the coder has completed the setup, they will hand off the computer to the designer, who will create sprites. These sprites should always be hand-drawn as custom sprites give an automatic 2 points, with up to 4, while default sprites, even with modifications, give a max of 2 points. Because of time, sprites should be designed to be simple to draw, usually simple shapes put together. My partner will cover this in the miscellaneous section.

Each game needs a user-controlled (UC) and an autonomous sprite. The UC sprite is usually the game character, while autonomous sprites are the things to collect/avoid in collecting/avoidance games. For example, in the
Cells ‘R’ Us game, the UC sprite would be the cell and the oxygen/glucose molecules would be the autonomous sprites. In this case, there isn’t a specific thing to avoid, but you do have to collect items.

**Step 2: UC Sprite**

If you look at the rubric, you will find that movement complexity is worth a total of 8 points. While diagonal movement is listed as complex, it’s better to use acceleration for UC sprites instead. This may sound complicated, but it’s actually very simple. A code example is shown.

In this code stack taken from the Cells ‘R’ Us game, we see the movement controls for the cell. First, we move the cell to a position on the screen, then show it. The forever loop following is where all the controls are managed. The if blocks check to see when the arrow keys are pressed which increments a velocity variable for the appropriate direction (x or y). Afterward, the sprite is moved by the velocity variable. This creates an acceleration effect because when you hold down an arrow key, the velocity increases and so does the amount stepped up. For deacceleration, each variable is multiplied by 0.95 to slowly reduce the number. These values can be tweaked to your liking.
In this specific example, the code to trigger the game end function is located in this loop. However, these blocks can be moved to a separate scoring loop.

**Step 3: Autonomous Sprites**

With autonomous sprites, movement complexity is simpler. While you could also implement some kind of acceleration, random movement is usually enough. In most collection/avoidance games, the items to collect/avoid use some sort of cloning. While I’ve heard about others having difficulty getting cloning to work, I’ve personally never had an issue.

Here is the cloning code for the oxygen molecule from the example game. In this startup code stack, we move the sprite off the screen and hide it. This is where the parent for the clones will be hidden. After that, we forever create clones every second. This next stack controls the movement for each clone when they are created. They show themselves, move to a random position on the top of the screen, then glide to another random position, where they will stay for a second then self-destruct. The ‘delete this clone’ block is very important, without it, you will get infinite cloning. After the movement stack, we have a separate scoring stack. This is because detection has to be in a loop, and you can’t have loops while waiting or gliding. The scoring stack of blocks detects if the clone is touching the player, then
increments the counters if so. Using this basic template, you can create all sorts of movement with the clones. For competitions, my partner and I default to having sprites move all the way across the screen from the top or the side and having collection/avoidance done this way. Here is an example. [https://scratch.mit.edu/projects/253876464/](https://scratch.mit.edu/projects/253876464/) (Not very sciencey and does not have a clearly defined outcome, but does have clones coming in from the side. Also has a level chooser which is nice)

**Step 4: End of Game**

When a player has won or lost your game, you’ll want to make that clear. If the coder has set up everything correctly, implementing the game end functionality should be super simple.

Here is the scoring code for the Cells ‘R’ Us game. First, we set up all the variables when the game starts. Next, we enter the forever loop which handles the glucose/oxygen to ATP conversion and the game end functionality. All you need to trigger a game end is to broadcast “Game End” and switch to the appropriate backdrop. The reason we did this instead of having a separate win/lose broadcast is that the only difference between a win and a loss is the end screen. If your game has other differences that require broadcasts, feel free to do that.

Besides just changing the background, you need to stop
gameplay, clear the screen, and have end of game options available. The coder should have already implemented end of game options in the beginning, so now, all you need to do is to stop gameplay and clear the screen. Luckily, this can all be done with a simple four block stack. This is all there is. Copy this stack into each autonomous sprite and each UC sprite, and that’s pretty much it. You can delete the “delete this clone” block in UC sprites which don’t use cloning, though it really doesn’t matter. If you showed variables during gameplay, you may want to hide them afterward depending on their function. Displaying a score, for example, would be useful to show on the end screen.

Additional Features

Along with this generic template, you may want to implement extra features such as level of difficulty. Having the difficulty increase/decrease as the game goes on is worth 2 points. This can be implemented by having a ‘level’ variable that which changes the speed or interval of collectible and avoidable objects.

If you have the schedule down and have plenty of leftover time, be creative. **Having a game stand out will boost the judges’ impression.** However, don’t do anything too crazy that may break your game, which may end up costing you more points.

Debugging

So something isn’t working. Don’t panic. This is perfectly natural and is bound to happen. During a competition, you’ll want to **let your partner know**, so you can have two people looking for the issue.
Before doing anything, ask yourself some questions.

- What did you expect was going to happen?
- What actually happened?
- What changes did you make since the last time it worked?
- How does this sprite interact with other sprites?

Thinking about these questions will help guide you to the solution.

After answering those questions, you’ll want to look through the code line by line. Most of the time, you’ll spot the missing/incorrect block right away. If not, look more closely. Pretend you’re the computer and think about each line. What does each block tell you to do? Sometimes, the issue isn’t even in the code. Take a step back to see if the code is even in the right sprite.

If the bug is not critical to the functionality of the game, you still can’t find it, and it’s a competition, you should ignore it and move on. You can come back to it later. However, if it’s not a competition or the bug is critical to the functionality of the game, you may want to debug further.

When looking deeper, you’ll want to play around with the code a bit. Take things out, change things up, and see what effect that has on the results. If the issue is that a block isn’t running, check to see if blocks around it are running. Using a ‘debug’ variable is very useful here.

If the issue persists after all that, get a friend or family member to take a look. Explain each and every line to your friend or family member like you’re teaching them how it works. I find that talking to a human works best, but you can talk to a stuffed animal or another object as well. This strategy is known as rubber duck debugging.

https://en.wikipedia.org/wiki/Rubber_duck_debugging
Grading a Game

After each practice, you should grade your own game or have someone else grade it for you to track your progress. Print out or look up a copy of the rubric and give it a go. Try to be as strict as possible to identify weak points in your game. If you’re ever hesitant on giving yourself a point, don’t do it. With super strict grading, try and get at least 80 points out of 100. You’ll also want to get as close to full points as possible on the Game Mechanics section. This way, you should get 80-90 points at a regular competition and basically guarantee a top 10 finish. At a national level, you’ll probably want to get at least 90 points with as close to max points as possible. But then, I’ve never been to nats so \(_-(- ITS)_/\.

As an example, the Cells ‘R’ Us game linked previously would score around 70 to 80 points depending on the judge. In this case with the rubric above, the judge was not very strict, giving us a score of 89/100. Taking a closer look, we see that we could have elaborated more on the instruction page and made certain things clearer. Because we ran out of room on the page, you could implement a page switcher with arrow keys within the help section to move Science of Theme to a separate page. We can also see that there is a lack of levels for this game with only one difficulty, costing us 2
points. Sprite orientation and environmental interactions are also extremely subjective and up to the judge, with the Rubric - Explained document saying “Sprite returns to original position or holds position given gameplay” and “Interaction is appropriate given rules and gameplay”. If there is time, you could probably add some mud, a rock, or walls depending on the kind of game, but it’s really still up to the judge. If your UC sprite has some sort of face, you may want to have costumes to switch the direction it’s pointing.

Preparing for Competition / Tips

Practice, practice, practice! Taking the time to familiarize yourself with Scratch will be invaluable to your team. Make lots of games of all different types, and just play around. Try making a space shooting game, a Pac-Man game, or even a quiz show game. Having that experience using Scratch will help you to develop a coding mindset and allow you to be more efficient while finding blocks to drag and drop. Getting together with your partner and simulating a real competition with a time limit and a theme will help a lot with teamwork and efficiency. If you can’t get together with your partner, having individual timed practices are a great way to practice by yourself. Brainstorming ideas for possible themes and game ideas to fit them is also a great way to prepare. Themes we encountered during competitions and practice include:

- Water
- Evolution
- Oxygen
- Cycles
- Color
- Weather
- Cancer
- Food Poisoning
- And so on...
The designer will also want to practice coming up with sprite ideas and drawing them as fast as possible.

Before a competition, you’ll want to prepare all your materials before you go to sleep. For Game On, this means pencils (multiple pencils), erasers, and a headset.

On the day of the competition, get there early if you aren’t traveling together with your team. Our coach usually asks us to arrive at least an hour before our first event or around when the first event of the day starts, whichever comes first. Having this extra time lets you check in with your coach and teammates, scope out the classroom locations, and get used to the environment. Knowing where your classrooms are at the beginning of the day will hopefully prevent you from getting lost/being late.

Misc. Tips and Additional Resources

Here are some additional tips/tricks that we’ve gathered over the season that don’t really fit into other categories.

Phenakism’s Way to Draw Efficiently

Hey everyone! I was Person-v132’s partner for this season, and I have a few more tips to add in order to get some extra points, and possibly impress a few judges. Judges will like your game much more if your drawings are nice, but that’s hard to do with all the other stuff you’re going to have to do in 50 minutes.
First off, always remember **KISS, keep it simple, stupid!** There's no use trying to draw a masterpiece with any of this, and the judges will hate your game if your game doesn’t work (even if your drawings are Picasso level). Stick with simple geometric shapes whenever you possibly can, and **avoid spending more than 10 minutes** on all designs.

Second, overuse the **reshape tool**. The time you can save by just using the reshape tool. Clouds, animals, water drops, and basically almost anything you might need are just a few drags away. Like in the pic above, I drew an amoeba by just reshaping a circle and placing a nucleus inside, which I hope generally looks like an amoeba.

Third, make the backdrop **“iconic.”** Do this by attaching simple icons to the backdrop which identify with the subject. For example, for a temperature game, I drew a mini thermostat next to the start button, a mini ice cube for the instructions, and kept the game screen a complete black (don’t add things just because you can, do it for points instead). If you can’t draw, try using a gradient instead. It will spice up your backdrops but without all the effort and time.

Fourth, make sure your sprites **have decent color**. Even though it's a little detail, it doesn't take much time to make sure that your graphics aren't an eyesore (i.e. if you're making a green character, don't make the others bright red). Take a good look at your sprites to see if you can see them clearly or if your eyes hurt when looking at them.

Finally, try to **avoid ever using the default library**. The judges will get royally sick of seeing the same default sprites in a lot of the lower-level comps, and you will stick out if you're in a higher-level one, and not in a good way. It may look easy, but if you really have to, at least edit it for a measly point. That’s probably all that could help you from me, so I’ll hand it back to Person-v132!

**Things to Keep In Mind**

When designing a game, you’ll want the game to be able to be won/lost fairly easily. Judges are most likely not going to be spending a ton of time
playing your game to completion, around 5-10 minutes max. You’ll want your game to be able to be played to completion several times in this time frame while still having a balanced game. Check the Rubric - Explained for more detail.

Documentation/comments are very important. It’s best to write comments along the way, but if you forget, that’s what the last 10 minutes are for. Always remember to clean up your code by right-clicking on a blank space and clicking “Clean up blocks”. This will give you points for coding efficiency.

Always keep saving! Computers/the internet used at competitions may be unreliable and crash. If using the online version and logged in to a competition account, click the “Save Now” button at the top right corner. If not logged in or using the offline version, periodically download your project to save it. You can delete the files afterward. To do so, go to ‘File’ > ‘Save to your computer’ on the Scratch website.

This guide should help you at the regional and maybe even at the state level. If you make it to nationals, you probably have a much better structure than this that works even better with more complicated game types. This guide will most likely not work at such a high level.

Other Game Types

   Shooting:
   https://scratch.mit.edu/projects/302862361/ - shooting tutorial

   Platformer:
   https://scratch.mit.edu/projects/287513869/ - example by my friend
   https://scratch.mit.edu/projects/34429446/ - platformer tutorial
Helpful Links

https://www.soinc.org/game-b - Links to rules/rubric
https://www.soinc.org/game - Official Game On pages
https://scioly.org/wiki/index.php/Game_On - Game On Wiki, lots of great info, highly recommend a read
https://scioly.org/tests/files/gameon_2019_b_ssss-carrot_notes.pdf - another SSSS guide, partially inspired this one, lots of great info, highly recommend a read

Conclusion

Thanks a lot for taking the time to read through my Game On guide. I hope you found it useful and wish the best of luck to your Science Olympiad journey. Please feel free to contact me or my partner with any questions, comments, or feedback you might have through my email (tomichen33@gmail.com), my partner’s email (jyehia06@gmail.com), or SciOly PMs. I will try to help as much as I can or direct you to someone more qualified.

Game On!