Carrot’s Game On Guide

Introduction

Hello! I am a rising senior at Mason High School and this is my guide for the Scioly event, Game On. Throughout the 2017-18 season, I have achieved a solid understanding of the event and hope to pass on my knowledge to others, as it will be leaving from Division C for the 2018-19 season.

The objective of the Scioly event, Game On, is to create a game in 50 minutes based off of a scientific theme, as well as a game type (2017-18) using the coding program, Scratch. This guide is meant for those who understand the basics of Scratch and who have read over the rules and rubric for the event.

In this guide, I will often be referring to the User controlled sprite as UI and the Autonomous sprite (the ones that the player doesn’t control) as AI. I will also be referring to the impression of the game as some sort of value, increasing with unique items within the game. This is due to the rubric having two extremely vague sections, each worth 4 points: “Overall impression of the game” and “Originality of the game.”

The Basics of Game On

When first experiencing Game On in any setting, one might wonder how two people can simultaneously work on the game, as there is only one computer, one mouse(usually), and one keyboard. Well, here’s what worked for me and my partner:

The Coder:

The coder is the one responsible for all the coding for the competition. They are the one dragging and dropping blocks within the program and should try to always be doing something at any given time during a competition. Game On is a battle with time; the more effectively you work, the greater success you will achieve.

The Designer:

The designer is the one responsible for the idea of the game as well as the scientific topics. Of course, the coder is allowed to also help with this process but it is not recommended as it would most likely lower the efficiency of both members. The designer should have a solid background in all fields of science, especially biology, in order to incorporate scientific topics into the game.

Competition - Quick Summary:

Immediately, the coder should start on the setup of the game (later to be explained). At the same time, the designer should begin brainstorming the game and telling the coder what type of movement for the UI and the AI sprites (this will be covered over more later). This process is basically continued for the next few minutes. While the coder is coding the game, the designer will also be writing the instructions and drawing sprites on the plan paper.

After this, either player would draw the sprites, background, and other visual aspects in the game. The designer would then type the instructions and set up the text for the win/loss screens as well as the start screen and game title.

Finally, both participants would work together to decide on background music, in game sounds, as well as type up the comments for the code.

The Setup

This section will cover how to carry out 'The Setup' for a Game On game. For almost every single game you make, the coder would have to do this as soon as the timer begins for the event. This section should be practiced by the coder to allow for them to carry out this section as quickly as possible.

Step 1: Create the Backdrops

By this I don't mean draw out all the backdrops. I just mean have each of the 5 backdrops created. Blank backdrops are completely fine for now.

Start - This is the screen that the game begins on when you press the green flag.
Instructions - This is the screen that appears once you click the instructions button. It will contain your instructions for the game.
Play - This is where you actually will play the game. It will appear once you hit the play button.
Win - This is the screen that appears if you win the game.
Lose - This is the screen that appears if you lose.

(Note: Usually if it is a two player game the Win/Loss screens would be replaced by Player 1 Wins / Player 2 Wins screens.)
(Another Note: I usually would color the Win backdrop green and the Lose backdrop red as soon as I make them.)
(Another Another Note: If it is a Maze game, draw out the maze ASAP. Will later be explained in game types.)

I've seen some teams not include a start screen and just have the game start on the instructions screen, therefore only have to code for the play button and not an instructions button. However, the Game On Rubric states that "buttons are used to access other screens/options." You may be able to get away with this at invitationals but I would highly recommend including the Start screen. It also helps build up the grader's impression over your game.

Step 2: Code the Buttons

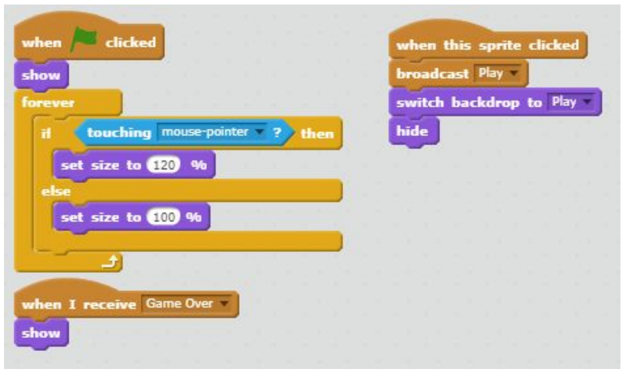
There are two buttons (usually) that the coder would code for in Game On during the setup: the Play button and the Instructions button.

Throughout various teams, some prefer to do things a little differently for the buttons.

- Some people draw their own buttons, and often times it is just a rectangle. I would highly advise NOT to do this. It is a complete waste of time and does not add much impression to your game. I always just use the “Button3” sprite from the scratch library (the grey colored one) and add text onto the sprite, either “Play” or “Instructions.” I guess if you had extra time at the end, you could redraw your buttons but if so, try to design them so that it fits the rest of your game.
- Some people use the term “Help” over “Instructions.” Yeah, go ahead if you want. There is nothing wrong with either way but be consistent throughout your game. Even the rubric knows it as “Help/Instructions.” The only downside of “Instructions” is that “Instructions” is a wee bit longer but with decent typing speed this shouldn’t really add any time into making your game.
- Some people (probably only my partner and I) make the button get bigger if you are hovering over it. This is a personal preference that I have developed over time which takes like 10 seconds to code but adds 2 forever loops into your program. I do this because it kind of looks cool and would stand out from other teams’ buttons even though it doesn’t add any extra points, maybe adds a bit of impression.
  (Note: I will be using this in the button code picture.)
- Some people have sprite changes on their buttons. Why?

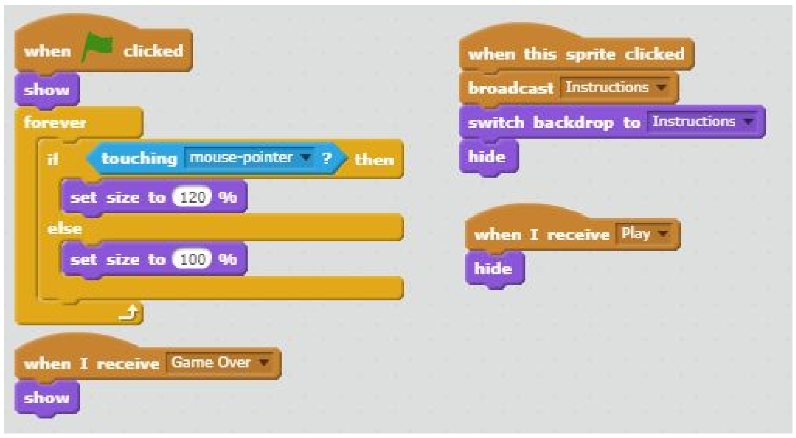
I will now put in an image of the code I usually have for both buttons during the setup.
(Note: Game Over code can be done either now or later, it really doesn’t matter)

Play Button:



```
when ⚑ clicked
show
forever
  if <touching [mouse-pointer ▼] ?> then
    set size to (120) %
  else
    set size to (100) %

when I receive [Game Over ▼]
show

when this sprite clicked
broadcast [Play ▼]
switch backdrop to [Play ▼]
hide
```

Instructions Button:

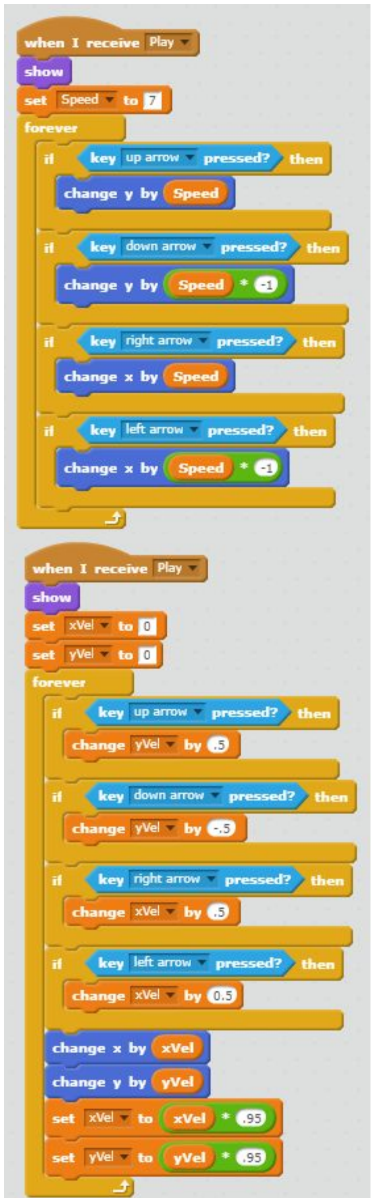


```
when green flag clicked
show
forever
    if touching mouse-pointer ? then
        set size to 120 %
    else
        set size to 100 %

when I receive Game Over
show

when this sprite clicked
broadcast Instructions
switch backdrop to Instructions
hide

when I receive Play
hide
```

Types of UI Movement

This section will cover the main types of UI movements that are commonly used in Scratch games.

(Note: WASD can be use d in replace of Arrow Keys based off of personal preference and if the mouse will be used in the game.) (Another Note: The numerical values I use for incrementing and initializing was just found by personal preference. Do whatever feels smooth for you.)

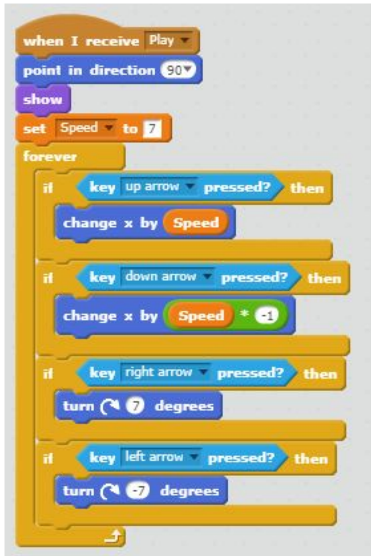


```
when I receive Play
show
set Speed to 7
forever
    if key up arrow pressed? then
        change y by Speed
    if key down arrow pressed? then
        change y by Speed * -1
    if key right arrow pressed? then
        change x by Speed
    if key left arrow pressed? then
        change x by Speed * -1

when I receive Play
show
set xVel to 0
set yVel to 0
forever
    if key up arrow pressed? then
        change yVel by .5
    if key down arrow pressed? then
        change yVel by -.5
    if key right arrow pressed? then
        change xVel by .5
    if key left arrow pressed? then
        change xVel by 0.5
    change x by xVel
    change y by yVel
    set xVel to xVel * .95
    set yVel to yVel * .95
```

Arrow Keys - No Velocity:
This is probably the most basic movement a user controlled sprite could have. Up arrow moves up, down arrow moves down, right arrow moves right, and left arrow moves left. The use of the speed variable is to later on make it simple to add in a speed change for the user controlled sprite, just by changing the speed variable.

Arrow Keys - Velocity:
This is an extension of the previous type of movement. It adds in both acceleration and friction to the game. The change in x velocity and y velocity variables are what allow the sprite accelerate and decelerate. The use of the 2 set blocks at the end allows for the deceleration of the sprite.

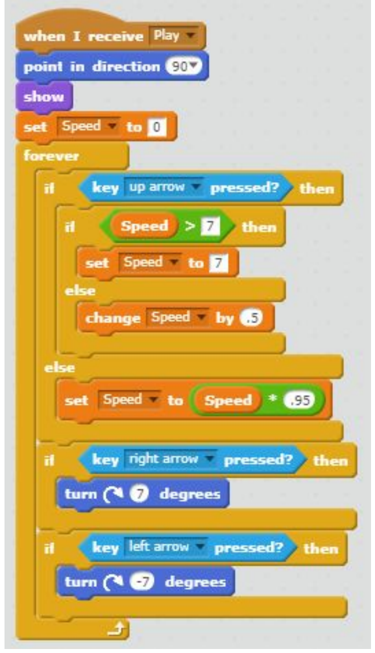
Car Controls - No Velocity:
This control type, or car controls, is controlled by pressing up arrow to move forward, down arrow to move backwards, right arrow to turn right, and left arrow to turn left. It is often used when the sprite you are making is from a top-down perspective.



```
when I receive Play
point in direction 90
show
set Speed to 7
forever
    if key up arrow pressed? then
        change x by Speed
    if key down arrow pressed? then
        change x by (Speed * -1)
    if key right arrow pressed? then
        turn ↻ 7 degrees
    if key left arrow pressed? then
        turn ↻ -7 degrees
```

Car Controls - Velocity:
This is an extension of the previous type of movement. It adds in both acceleration and friction to the game. The first first else is used to decelerate. The if-else speed > 7 test is used to make 7 the maximum speed that the sprite can move at, capping the speed.

(Note: This example of car velocity does not involve going backwards. It can be easily implemented if needed, though let's be honest, how often do people go backwards?)



```
when I receive Play
point in direction 90
show
set Speed to 0
forever
    if key up arrow pressed? then
        if Speed > 7 then
            set Speed to 7
        else
            change Speed by .5
    else
        set Speed to (Speed * .95)
    if key right arrow pressed? then
        turn ↻ 7 degrees
    if key left arrow pressed? then
        turn ↻ -7 degrees
```

Most of the times my partner and I would mainly use either Car Controls - Velocity or Arrow Keys - Velocity for our UI movement type. This is to allow for an overall smoother feel as well as sometimes being counted for points in complex movement of the UI. There are, however, exceptions to this. For example, often times in a maze game we would choose to use Arrow Keys - No Velocity as all the other types of movement in a maze feel really messed up as you would just keep crashing into walls over and over again and it would be hard to control.

In a note found at the beginning of this section, I said that WASD could replace the arrow keys, as they are basically the same exact thing. I preferred WASD as they felt more natural to me.

This also allows for using the mouse for things such as shooting. If you use arrow keys, using the mouse at the same time would feel rather awkward.
For 2 player games, I would usually make one of the user sprites move with the WASD keys while the other user sprite move with the arrow keys. You can choose whether you want car controls or not with one exception: 2 player racing games MUST use car controls. This is because you would usually take the direction the UI is facing and just have the bullet go in the same direction, instead of shooting using the mouse since there is not a mouse for both people.

Types of AI Movement

This section will cover the simplest types of AI movements that are commonly used in Scratch games. It will also cover a few very conditional AIs that are rarely used but could save the day when given a strange game type combination. The movement type(s) used should be decided by the Designer. Multiple movement types can be used within one game assuming that you have multiple AIs.

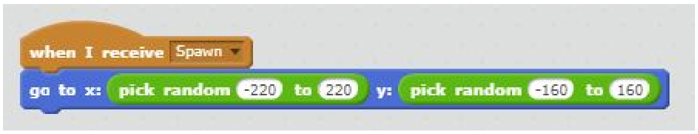
None of these AI Movements types are meant for competition use. They are meant to be improved and altered depending on the scientific theme given. If you use any of these types of AI movement, you will lose many points as none of these are very complex.

Creating your own unique autonomous sprites is a big part in preparing for competitions. The more you know the better! These are just a few extremely simple examples with one very situational one.

Follow - Direct:
This is very easy to code. It is a forever loop with a point towards UI and move. It is the simplest form of chasing the user that you could possibly get.

Follow - Glide:
This is very similar to the Follow - Direct code except you would have a glide to x of UI, y of UI instead of the point and move. It is another simple form of chasing.

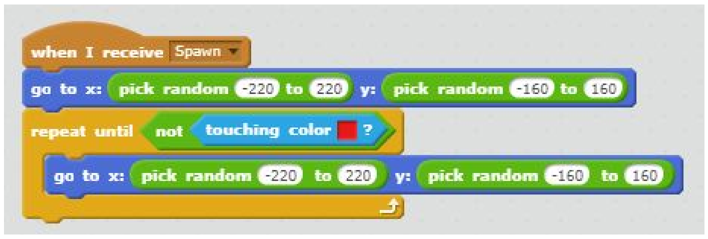
Spawn:
This code is extremely useful. I would say around 75% of the games I made this Scioly season involved a variation of this code. It just moves the sprite to a random location.



```
when I receive Spawn
go to x: pick random -220 to 220 y: pick random -160 to 160
```

- The reason why the values 220 and 160 are used is because the max range of the x is up to 240 and the max range of the y is 180. Using 220 and 160, you can ensure that the sprite will always be able to be seen and be on the screen.

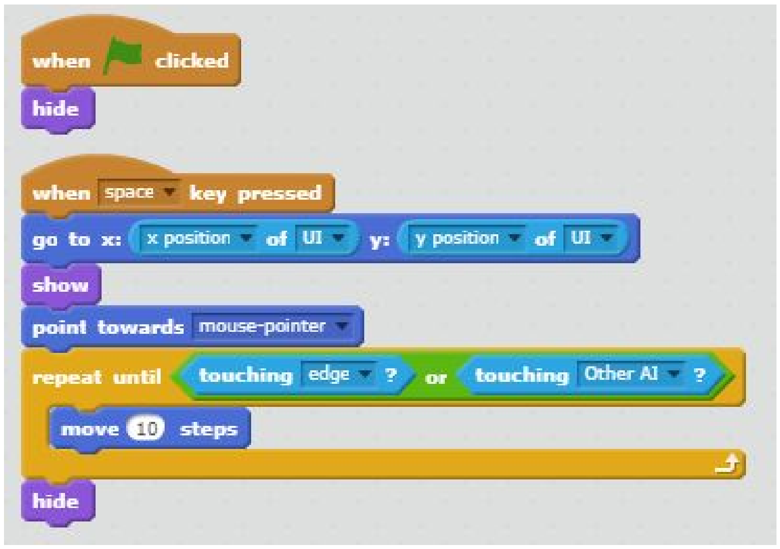
If you want the sprite to not touch a certain sprite/color, then do the following:



```
when I receive Spawn
go to x: (pick random -220 to 220) y: (pick random -160 to 160)
repeat until <not <touching color 🟥 ?>>
    go to x: (pick random -220 to 220) y: (pick random -160 to 160)
```

Bullet:
This is the code for a bullet in case you want to include shooting. There are two main forms of shooting: shoot towards mouse or shoot in the same direction that the UI is facing. In the following code I will be using shoot towards mouse.

Also, the way I do shooting is I have 1 bullet on the screen at the same time. If you want to, you can mess with clone shooting but I have very bad past experiences with clones and almost always avoid using clones during competitions.



```
when ⚑ clicked
hide

when space key pressed
go to x: (x position of UI) y: (y position of UI)
show
point towards mouse-pointer
repeat until <<touching edge ?> or <touching Other AI ?>>
    move 10 steps
hide
```

You can also add acceleration and deceleration to the bullet for more complex movement and a higher impression which I usually do when I use a bullet. To do that, in the repeat until loop you would have to increment a speed variable and then move by speed. You would also need to initially set the speed to 0 a value like that.

Maze Racing:
Ah, this sprite is very special to me as I came up with it on the day of nationals during the impound time after realizing that I was screwed if I got this deadly game type combination: Maze and Racing. This sprite is kind of like a pacman ghost sprite but it has more random motion. To make this sprite successfully, you have to draw the maze in a very specific way. This will later be elaborated on later.

(It seems that I have lost a code example for this. It is basically a sprite that turns either left or right every time it hits a wall.)

Game Types

This section will cover all six of the game types found in the 2017-18 Game On rules.

Collection – the user controlled sprite is involved in the collecting of objects to complete the objective of the game.

Nearly every game over this season that I made had some sort of collection within it. Collection is a very simple thing that you need to collect objects in order to beat the game. To do this, just use a collision test with an AI and the spawn code and poof, you got yourself a collection game. Of course, it still needs to be made more complex but that is the basis of a collection game.

Maze – the user controlled sprite must navigate through a series of static obstacles, borders, boundaries or lines to complete the objective of the game

Yeah, a maze game is just a game where you are in a maze. Try not to stretch this, such as by just having a few obstacles, just in case a grader tiers you for the game not being a maze. Yeah, just make a maze.

Oh boy, it's already time for me to reveal the easiest way to make a maze without sheer memorization and line drawing. My hidden OP throughout the season…

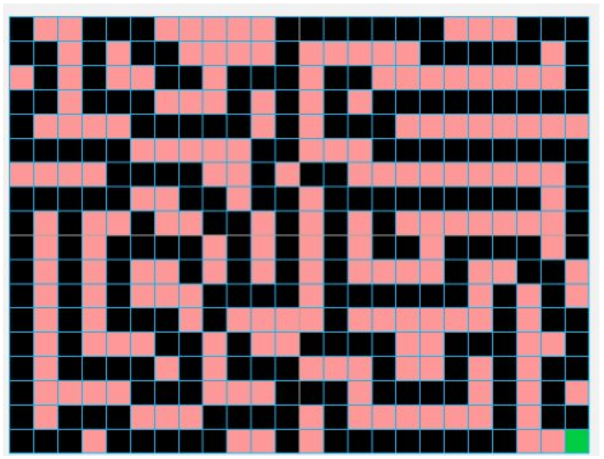
Go to the Scratch backdrop library and select the "xy-grid-30px" backdrop. It is basically your maze canvas.

By default, the backdrop begins in vector mode. While on this mode before switching it over to bitmap mode, using the fill bucket will allow you to choose the background color for the entire maze, basically the color that will act as the walls within your maze. Once you have done this, switch the backdrop to be in bitmap mode. Using the fill bucket now, you will be able to fill in the individual squares that make up the path of a maze.

Tips:

- Usually always have distinct starting coordinates for the UI sprite
- If you are trying to get to the end of a maze, define the end of the maze using a different color patch of squares. You can then do a simple color collision test to see when the player has reached the end.
- When drawing the maze, try to use up as much space as possible. If your path has lots of touch corners, so that in a 2x2 area, 2 squares are touching at the corner, your maze will look more visually appealing and complex.

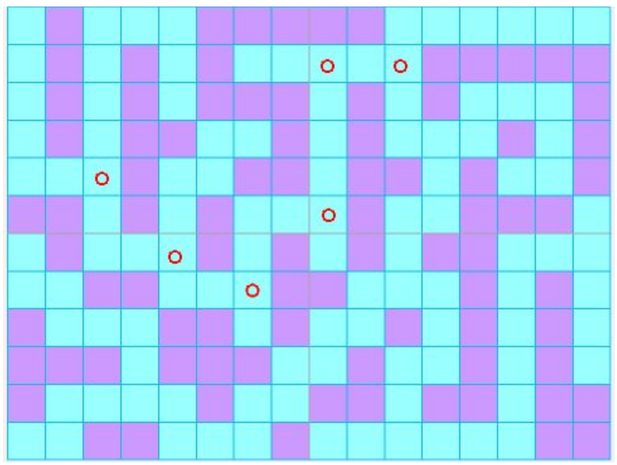
Here is an example of one of my mazes:



In this maze, the colors used could be better but eh, it gets the job done. The black is the path, pink is the walls, start at top left, end at bottom right. This maze is also done using the 20px grid, that's why it might look a bit more complex. The next maze is over a 30px grid, so it would better represent what you make.

The biggest downside of using this way to make the maze is that most of the sprites have to be small.

Now for the Maze Racing combination:
First, choose a distinct location that the Maze Racer AI will start at (if you do not know what the Maze Racer AI is please refer to the AI movement section). You should then make a path that uses T shapes to lead the AI to the end of the maze. This way, once the AI passes the through the T intersection, it will no longer be able to go back through the stem of the maze.



Here is an example of a maze that incorporates the T shape interceptions. Blue paths, purple walls. In this case, the AI would start at the top left and move its way to the top right, not being able to go back once it reaches a T intersection. The T intersections that the AI will pass through to get to the end are marked with red circles. And thus you pass through the deadly game type combination of maze and racing.

Avoidance – the user controlled sprite must avoid moving autonomous sprites to complete the objective of the game State

Avoidance is pretty straightforward: avoid the moving AI. Please don't have the AI that you are avoiding not moving, that's no fun at all.

A common type of game that I used to make a lot, especially in both the 2015-16 and 2016-17 seasons are chase games - basically a cat and mouse game. You collect something that scores you points while avoiding something that is chasing you, losing points / lose the game on contact. Have a certain number of points once the time is up. This game was a decent scoring game that I basically spammed the entirety of those two seasons. It is definitely not a top scoring game as it is not creative or original but the basic concept of the game still can be used.

<u>STATES GAME TYPES:</u>

Shooting – the user controlled sprite must shoot or direct an object(s) during the game to complete the objective of the game

Again, like the other game types, I would try to make a game that actually involves shooting rather than just directing an object. This way there is no chance that you will be tiered.

Shooting is, yeah, shooting. You are a sprite who shoots something at an AI. Simple enough. The way I would usually make a shooting game is basically the chase game that I mentioned earlier but instead of collecting something I would shoot the object. Then again, I am not the Designer for the game, so this is not my area of expertise.

Racing – the user controlled sprite must complete the objectives of the game before the autonomous sprite does

Racing is where you compete against an AI to achieve the object of the game faster. Often times my racing games involve a chase game where the AI chases after the collectable instead of the UI. See now how useful branching off of the chase game is?

This previous season, 2017-18, there was a huge discussion over 2 player racing. Granted this is only a nationals game type but it showed up at the 2018 MIT invitational. In this invitational, all but 4 teams where tiered for not following the game type. This is because most people interpreted a 2 player racing game as player vs player where it really had to be player vs player vs AI or something like that because the AI still had to race the UIs.

Building – the user controlled sprite must be involved in the assembling of smaller parts or components to complete the objectives in the game

I have to admit, even up to this day I still do not have a concrete building game template. The building game type was a widely discussed topic on the 2017-18 Game On forums with many people doing building many different ways; however, my partner and I never really had a concrete way to do building but somehow never ended up having to do it in a competition.
Oh, and when you do building make sure the UI is doing the building, not the other sprites. This is very important.

NATIONALS GAME TYPES:

Combination of any two of the previous game types

Two player game of any one of the previous game types

I am not going to cover national game types too much because they are pretty self explanatory. One thing to remember is when in practice, don't give yourself game types that don't make sense together with the theme as from my experience all the themes are very reasonable and achievable with the game types.

Some tough combinations (from my experience in practice):

- Maze + Racing (covered over in this guide)
- 2 Player + Maze (hard to make the game balance)
- Building + Shooting
- Building + Racing
- Building + Maze

(I really struggled with building ;-;)

Points Lossed

This section will cover where most teams lose points in competition.
(Note: An explanation of the rubric can be found on soinc.org)

Game Mechanics:
When I talk to my peers about Game On, the first thing that always comes up is how many points did I lose on the left hand side of the rubric in the game mechanics section. Now looking back at it, I realize this was very silly of me. I should always be scoring 50/50 on the game mechanics section.

Where people usually lose points in Game Mechanics:

- The UC Sprite needs to have complex movement. Most grader's do not count diagonal movement to be complex movement, although the Soinc explanation says so. The best and easiest way to get these points is, by far, UC acceleration/velocity.
- Similarly, AI Sprite often are missing complex movement. I would also recommend acceleration/velocity for this, but you can explore other ways to get this point yourself, as it is rather wide.
- Make sure to have sprite collisions with the background. I often forget about this during competition, and remember it at the last second, scrambling to add something in.
- In the debriefing section, make sure that you have a game over. For some reason I've seen teams that say "Click the Green Flag to Play Again." This is unacceptable. All you need to do is broadcast a Game Over message and then for all your game sprites, hide them and stop their code. For the buttons, show them. Done. Easy.
- Again in the debriefing section, make sure that your variables are hidden and shown when they are supposed to, not only the sprites.
- In the documentation section, make sure you have code for EVERY SINGLE SPRITE. Even if multiple sprites are the same, just copy and paste the comments. Also make sure to not BS your comments and actually write things that help the grader understand your code. Oh, and if you didn't know, you add comments by right clicking and selecting "comment."
- In the Code Organization section sometimes points are lossed for the Coding is Efficient. Depending on how well the grader knows what he/she is doing (so if they actually know what efficient code is or not), will cause your points here to change. Generally people would only lose points in this section in big invitationals (such as at MIT) as people know more about coding there. For most things, a right click and "clean up" will do.

Game Play

Lots of the game play section is up to how the graders' grade. You have been warned.

- Nearly every game that I have ever made, I have lossed points in the "Science of theme" section. To get full points, you need to come up with multiple scientific processes that support the theme given, explain what they are / what they do, and explain how you incorporated them into your game.
- Graphics is a big oof for me. Yeah, draw good sprites and backdrops. What more can I say? Oh, and make sure that your sprites have meaningful sprite changes. Oh, and make sure that your backdrop makes sense to the theme and your game. Don't just have some sort of generic backdrop.
- Sounds is a section that I always sacrifice points in. Myself, personally, I just use the sounds found in the Scratch Library. I find that everytime I want to make custom sounds, my microphone doesn't work or picks up too much noise. When adding sounds, make sure they make sense within the game and it is best to have them add on to the science of the theme.
- Play Balance is another section that has a lot up to how the grader grades. The way I get the levels point is by having a speed change in the AI once the player reaches a

certain point count or a certain amount of time has passed. I then specifically mention that this is a level change in the instructions. Also, make sure you have both a Win/Loss condition as well as a points system within the game.

- The “Overall Game” section is every Game On member’s worst nightmare. It’s either really good or really bad. I talk about some ways to increase the impression of the game in this guide and this is what it is for. Usually if you have a high scoring game, the Overall Game section will also have a higher score. Also, the best way to increase the score in this section is to be unique. Know that the grader is grading numerous games one by one. Try to make your game stand out; maybe interpret the game theme differently while making sure you don’t get tiered.

Preparation

(Another Note: this is all my opinion on how Game On members should prepare based off of my past experiences)

How to Start Preparing

(Note: both the Coder and the Designer should work through these steps separately)

Many people I know have said that anyone can pick up Game On in an hour. They called it a meme event and for many teams it was just another one of those events that you don’t expect to do well. What people don’t realize is how competitive Game On gets with the point by point grind. I do admit that the Game On rubric is not very good as many parts are very vague and up for interpretation, however, the rules might slightly be changed for the 2018-19 Scioly Div. B season.

Once you tell yourself that you are going to do Game On AND that you will not treat it like it is the dumbest and easiest event (although it might be) you should learn the scoring rubric. By competition time, you should be able to know where all 100 points come from AND how to get them. This will allow you to decide which points you are going to sacrifice in order to get everything done in the 50 minutes allocated, which comes back to the coder having to be able to work as fast as possible. I am sure that most of the competitive Game On duo’s out there are able to get all 100 points within a 3 hour span, but how many can pull off 85+ points (decent score) in just 50 minutes?

I am not going to cover specifics over how to get each individual point out there as there are numerous resources out there, whether it be on the Scioly wiki or the forums, that contain this information.

“I’ve learned the rules. Now what should I do?”

Good question. You should make a chase game. “The kind of game that you told us not to make?” Yes. A chase game hits most parts of the rubric and can easily achieve an 80+ score in 50 minutes if done properly.

I do not expect your first game to be godly or perfect or even hit all the points. I just want you to try to learn the code yourself. If you make it yourself without the use of any outside resources, you truly learn the code. Do not become one of those people who memorize code line by line instead of modeling the behavior and figuring out how the code should be written. If you learn by yourself, you can come up with more creative things you can do on the spot rather quickly.

After you have made your first game, the next thing to do is look at someone else’s game (preferably a high scoring one) and read their code. You are doing this not to memorize blocks of code but to understand how they did things and to see if you did something in a different way that they did the same exact thing allowing for you to analyze the efficiency of your own code. This way, you are looking back at your own code while also improving your coding skills and your code’s efficiency.

Some time later, preferably not the same day, remake the chase game you made adding is small variations if you would like. Try to incorporate some of the things that the game you looked at had. This helps you use the skills that you have seen in work by others and apply it to your own game. After this is done, grade your game.

One thing I found when I had other Scioly team members grade my games is that their interpretations on the rubric is much looser than what the intentions of the rubric are. I’d often have games graded by friends score in the 90s while in competitions similar games would only score in the 70s. You know the rubric the best so you should grade your own game and trust the grading. Oh, and if you are ever iffy if you got a point or not, just say you missed the point.
This is just the very beginning of your Game On experience. Now I will be covering how you should prepare in general versus where to begin preparing.

How the Coder Should Prepare

A coder’s main job is to code, not to design. Although the coder can help out with the design of the game, they should avoid this and focus on the code of the game to most efficiently use the 50 minutes given for this event.

A coder should have a folder (or Scratch.mit account) that contains each individual coding bit that they use within the season. The folder should contain things such as:

- The Setup
- Each of the movement types, INDIVIDUALLY (this is key as these files will only be used for reference by yourself)
- Any games you make during the season(s) with the dates that they were made on

- Any additional gimmicks that are made throughout the season(s) (ex. Health bars)

The coder should develop a habit of trying not to refer to this folder unless preparing for an upcoming competition. This way the coder can develop the skills required in an actual competition, dragging and dropping the blocks without any outside resources.

The coder should also try to develop as many additional gimmicks that they can think of throughout the season. You never know what you might need given a certain theme and game type combination.

Individually, the coder should choose two or three game types and make a game based off of it, without a theme. Just use basic shapes, such as circles and squares, for sprites. The coder only needs to code the stuff that actually goes on when the game is being played; no need for instructions, win/loss, commenting, sound, etc.

When preparing with their partner, the coder should work at a moderate pace trying to make sure that everything should work. Test the program often. If something doesn't work, look through the code together with the designer in order to have a better success debugging the code. Unlike the individual preparation, the coder should code for a full game rather than just the play mechanics.

How to Code
In this subsection I will explain the basics of Scratch coding incase you are new to it.

The Sections in Scratch
- Motion: This section contains all of the blocks that move the sprite that you are working with.
- Looks: This section contains all of the blocks that change how the sprite looks or the backdrop costume.
- Sound: This section contains all of the blocks that add sound effects to it.
- Pen: This section contains blocks that allow you to draw onto the backdrop.
- Data: This section contains blocks that work with variables. Variables are used to hold data (usually numbers for us), just like in Algebra.
- Events: This section contains blocks that start and end chunks of code.
- Control: This is probably the most important sections out of all of them. It contains all the logic things such as testing and looping.
- Sensing: This section contains blocks that test to see what is going on. Most of the blocks here are booleans (things that are true or false) which are depicted in a "?" in the blocks name.
- Operators: This section contains all your math operations and math functions.
- More Blocks: This section contains lets you make your own block chunks and use it multiple times. You most likely will not need to use this section while making a game in 50 minutes; however, you can experiment with it if you want.

Learn to Code
Now that you know what all the sections are in Scratch and have looked through some of the code blocks you may be wondering how to start putting the blocks together and begin to code…

The first thing you will want to do is create a sprite; it doesn't have to be anything fancy. Once you do that, select the sprite. The blocks you put on the right will now only be for the sprite selected. If you would like to code for the whole program and not that specific sprite, select the backdrop and code there.

Next, drag the "When Green Flag Clicked" block from the Events tab. Follow this my some kind of loop as some kind of move. And there you go! You now have a moving sprite.

To learn more about coding, I would recommend checking out the "Catch Game" found in the help section (Click the "?" in the top right hand corner of the screen) as this is the very basics that demonstrates a loop and a test. Oh, and when I say test, I mean the "If" block that can be found under the Control tab.

Debugging
When starting out in Scratch, or any type of code language, debugging is a huge part of learning how to code. By the time the season is over, you should be a master at coding and rarely have to debug in competition, even if you are trying something new for the first time. You should be able to write code with little to know error as Game On is all about time management and cramming points in.

Hands down, the best way to debug your code is to reread it over and over again. Understand that your code is running line by line. Use this knowledge to see where the code is breaking; look for where the code got to by knowing what has run and what is not running.

Some things to keep in mind while debugging:

- Make sure you don't have collision sprites between 2 sprites in both bodies of code. Code is never run simultaneously in Scratch, it is always line by line. Therefore, if you have 2 collision tests for the same test, only one will happen.
- I always avoid clones. They hate me. If you want, you can figure out how to use them without getting infinite cloning or glitches.
- Make sure to check that the code section you are looking over is under the sprite it is meant to be under. You don't want to know how many times I have made this silly mistake.
- Make sure when drawing backdrops that you don't use vector mode. Vector mode, for some reason that I do not know, is very buggy for backdrops. It is perfectly fine for drawing sprites, though.

During competition, if I come across a bug, I immediately notify my partner and ask them to help me look over the code while I am also looking for the bug (Remember: your partner should also learn/know the basics of Scratch coding). Two times the efficiency. If you still can't find the bug, scrap the idea if it is small. Your goal is to waste as little time as possible.

Outside of competition, NEVER leave a bug you come across unsolved (Ha. I'm a hypocrite. I just ditched clones. I've spent hours upon hours looking in to the clone bugs and still have yet to find a solution). If you can't find the problem yourself, get some help. Ask your partner, ask others in the event, ask anyone who knows how to code. Shoot me an email if you need to.

How the Designer Should Prepare

As previously stated, the designer should have a solid understanding of many scientific topics. Preferably, the designer should have taken most high level science classes within their school curriculum. I do realize that moving this to division B means that students will not know as much information over scientific topics unlike those in division C but I expect that the topics given will be more reasonable for division B competitors.

The designer should create an organized document that holds a list of possible game themes that they come up with. They can also ask others for possible game themes.

Once a solid list is compiled, the designer should then choose a few topics at a time and plan out a game for them, without being given the game types. Make sure to include at least 4 scientific concepts within the game, use the internet if needed. When the designer finishes planning out a game, they should list off to the side all the game types that the game they created fits. This process should be repeated over and over again, that is the way to prepare as a designer.

Remember, an event supervisor would never give you a theme with a game type that does not relate to it in any way, probably. When one writes out the game types for the games you make, there is a high probability that with the theme you are working on an event supervisor would choose one of the game types that the designer listed.

When preparing with their partner, the planner should write on a piece of paper of every scientific concept they know over a topic before planning out the mechanics of the game. They should then convey their ideas to the coder for their opinions on the game they designed. In this event, communication is key; the faster the coder understands the game that the designer creates the more successful the duo will be. The designer would then draw out the sprites and write the instructions. Make sure to read over the instructions multiple times. They must be clear and understandable for the grader. Once done with this, watch as the coder codes making sure to point out any bugs or mistakes in the code that you might see.

Welp, that’s about it for this guide. I wish you best of luck in your time spent on Game On.

Feel free to contact me through Scioly private messages or through email (nsong2001@gmail.com) over any questions you might have for me over Game On or anything else, though I may not always have an answer.

Last Edit: 06/14/2018