Advanced Scratch Programming - learn to design programs for challenging games, puzzles and animations

Abhay B Joshi
Introduction

Background

The idea of using computer programming as a medium for learning was first proposed by Seymour Papert in his ground-breaking book “Mindstorms” back in the 1980s. Dr. Papert also designed the Logo programming language specifically for this purpose. For a variety of reasons this idea remained an esoteric one and did not become main-stream until almost the turn of the century. The last decade and half has seen a dramatic change in this situation. Today, no one questions the idea anymore. The benefits of learning programming and computer science concepts well before college – even in elementary grades – are well-understood. Everyone from the President of the United States to CEOs of major tech companies to school administrators are talking about teaching computational thinking to children and young adults. The focus is now on how to implement this idea in schools.

Here is a list of some of the amazing things that happen when children engage in computer programming:

- Children become active and creative learners, because they explore ideas through a hands-on activity with an infinitely powerful tool.
- They learn to think about and analyze their own thinking, because that is the only way to program computers.
- They learn to solve complex problems by breaking them into smaller sub-problems.
- They learn a new way of thinking (called “computational” thinking).
- In the world of programming, answers are not simply “right” or “wrong”; this prepares a child’s mindset for real-life problems.
- Children’s learning processes are transformed from acquiring facts to thinking creatively and analytically.

There are several books and papers on this subject of “computational thinking for children and young adults”. If you would like to get a quick overview of this subject, please check out my 2-page handout at:

http://scratched.gse.harvard.edu/resources/handout-learning-through-programming
About this interactive book

The choice of programming language is critical to achieve the intended objectives of teaching CS to beginners. In this book we use the Scratch programming language. Scratch is an entertaining and powerful language, and yet it is easy to learn. It is known as a “low floor and high ceiling” language – it allows the learner to build his/her vocabulary without getting mired in the complexities of syntax and grammar.

There is a lot of material on Scratch Programming on the Internet, including videos, online courses, Scratch projects, and so on, but, most of it is introductory. There is very little that can take students to the next level, where they can apply their Scratch and CS concepts to exciting and challenging problems. There is also very little material that shows students how to design complex projects, and introduces them to the process of programming.

This book is meant to fill these gaps.

In short, this book is for students who are already familiar with Scratch: its various commands, its user interface, and how it represents a variety of CS concepts such as, variables, conditional statements, looping, and so on. The book does not attempt to teach these concepts, but, it does provide a quick introduction to each concept in the free Supplement to the book, as explained below.

I call this an “interactive book” because it is something between a traditional book – which is static and passive – and a fully interactive online course. It does look like a book: it has a series of chapters, diagrams, a lot of text, etc. But it also contains links to online Scratch programs, code snippets, references, which the reader is expected to click and explore to fully benefit from the ideas presented.

Supplement for CS concepts

The projects covered in this book are all based on a variety of CS and Scratch concepts. The book does not attempt to explain these concepts. But, I have prepared a supplement that provides a brief description of each of these concepts. It is NOT a rigorous and comprehensive explanation of concepts, but only a quick summary.

This supplement is available for a free download at:

What is in the book?

I have organized the book as a series of independent Scratch projects – each of which describes how to design and build an interesting and challenging Scratch program.
Each project progresses in stages – from a simple implementation to increasingly complex versions. You can read these chapters in any order you like, although I have tried to arrange the chapters in an increasing order of challenge.

Programming is a powerful tool that can be applied to virtually any field of human endeavor. I have tried to maintain a good diversity of applications in this book. You will find the following types of projects:

- Simple ball games
- Puzzle games
- Memory games
- Science simulations
- Math games
- Geometric designs

**Learn the concepts**

As the experts will tell you, concepts are really understood and internalized when you apply them to solve problems. The purpose of this book is to help you apply Scratch and CS concepts to solve interesting and challenging programming problems. Every chapter lists, at the very start, the Scratch and CS concepts that you will apply while building that project.

**Learn the design process**

Besides these technical concepts, you will also learn the “divide and conquer” approach of problem-solving. This is a fancy term for the technique of breaking down a bigger problem into many smaller problems and solving them separately one by one.

You will also learn the “iterative design process” for designing programs. This is another fancy name that describes the idea that something complex can be designed in a repeated idea -> implement -> test cycle, such that in each cycle we add a little more complexity.

You will also learn a bit of “project management”. Project management helps you undertake a project, such as creating a complex program, and complete it in a reasonable time, with reasonable effort, and with reasonable quality. It involves things such as planning tasks, tracking their progress, etc.
Audience for the book

The book is intended for students who are already familiar with Scratch. The level of challenge is tuned for middle- and high-school students, but elementary-school students who have picked up all the concepts in an introductory course might also be able to enjoy the projects presented in this book.

The book would be a great resource for teachers who teach Scratch programming. They could use the projects to teach advanced tricks of programming and to show how complex programs are designed.

Finally, the book is for anyone who wants to get the wonderful taste of the entertaining and creative aspect of Computer Programming.

Hardware and software

To write programs described in this book you will need Scratch 2.0. There is an online version of Scratch 2.0 at http://scratch.mit.edu which you could use. You could also download the offline version of Scratch 2.0 available at https://scratch.mit.edu/scratch2download/. The offline Scratch editor works on most Windows, Mac, and Linux platforms.

Abhay B. Joshi (abjoshi@yahoo.com)
Seattle, USA
August 2016
Author's Background

As a freelance teacher (since 2008), Abhay’s area of interest has been teaching “Computer Programming as a medium for learning” and he has been teaching Scratch regularly to elementary, middle, and high school students in Redmond and Bellevue, WA. He has also been teaching 7th and 8th grade students at Aksharnandan School in Pune, India every summer since 2008. In 2011 Abhay co-authored (with Sandesh Gaikwad) two books on Logo Programming. Abhay has written several articles to promote computer programming, and has conducted teacher-training workshops to encourage aspiring teachers to experiment with this idea.

Abhay has been associated with the Software Industry since 1988 as a programmer, developer, entrepreneur, and a teacher. After getting an MS in Computer Engineering from Syracuse University (USA), he worked as a programmer for product companies that developed operating systems, network protocols, and secure software. In 1997, Abhay co-founded Disha Technologies, a successful software services organization.

Programming remains one of Abhay’s favorite hobbies, and he continues to explore the “entertaining, intellectual, and educational” aspects of programming.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Me First</td>
<td>XIII</td>
</tr>
<tr>
<td>1. Game of Bricks</td>
<td>1</td>
</tr>
<tr>
<td>2. Square Designs</td>
<td>19</td>
</tr>
<tr>
<td>3. Optical Reflection</td>
<td>41</td>
</tr>
<tr>
<td>4. Ping Pong</td>
<td>57</td>
</tr>
<tr>
<td>5. Digital and Analog Clock</td>
<td>75</td>
</tr>
<tr>
<td>6. Recursive Designs</td>
<td>103</td>
</tr>
<tr>
<td>7. Picture Cards</td>
<td>125</td>
</tr>
<tr>
<td>8. Tower of Hanoi</td>
<td>141</td>
</tr>
<tr>
<td>9. Connect Four</td>
<td>165</td>
</tr>
<tr>
<td>10. Guess Arithmetic Operation</td>
<td>177</td>
</tr>
<tr>
<td>Appendix</td>
<td>196</td>
</tr>
</tbody>
</table>
**Game Description**

This is a game you play with a ball and a flat paddle. A number of bricks are lined up at the top of the screen. As the ball bounces up and down you use the paddle to hit the bricks and score points. The ball must not touch the ground: after 3 such touches you lose the game. If you hit all the bricks you win the game. You can control the difficulty level of the game by changing the speed of the ball.

Do you want to check out a working Scratch version of this program? Click on the image below (or the URL just below it). I encourage you to explore the program and its various features. But, don’t look at the Scratch scripts yet; we want to design this program ourselves!

**How to play the game:**

1. Click on the “Green flag”: everything is reset to the original state.
2. Set ball speed using the slider.
3. Press SPACE BAR to start the game.

Link: https://scratch.mit.edu/projects/104716301/
Scratch and CS Concepts Used

When we design this program, we will make use of the following Scratch and CS concepts. I assume that you are already familiar with these concepts. If not, or if you want to brush up on these concepts, you should refer to the free downloadable supplement to this book at http://www.abhayjoshi.net/scratch/book1/supplement.pdf.

Main concepts:
- Algorithms
- Backdrops - multiple
- Concurrency - running scripts in parallel
- Concurrency - race condition
- Conditionals (IF)
- Conditionals (Wait until)
- Costumes
- Events
- Looping - simple (repeat, forever)
- Motion - absolute
- Motion - relative
- Motion - smooth using repeat
- Relational operators (=, <, >)
- Sensing
- Sequence
- Stopping scripts
- Synchronization using broadcasting
- User events (keyboard)
- User events (mouse)
- Variables - numbers
- Variables - as remote control
- Variables - properties (built-in)
- XY Geometry

Additional concepts (for the advanced version):
- OOP - creating instances using clones
- Random numbers
**High Level Design:**

Let’s take a look at the main screen of the game and try to point out the different pieces.

![Diagram of game screen]

The order in which we should work on these different pieces of the program is really up to us. It probably makes sense to first get the ball bouncing around the screen. Then, we will add a paddle that can be moved by the user. The ball must bounce off the paddle. Next, we will add the bricks which must disappear upon touching the ball. Next, we will add the idea of “number of lives” to the game. Finally, there are a few little things that will wrap up the game, such as variables to count things, controlling the speed of the ball, etc.

So, let’s get rolling with these various ideas one by one. Be sure to try writing your own scripts for each idea before looking up the “Solutions” section.

**Initial Version**

In the initial version of the program, we will work on the following feature ideas:

- Get a “ball” sprite and make it bounce freely. After pressing SPACE BAR the ball should start bouncing around, primarily in the vertical direction.
- Add the paddle. The paddle should move left-right only and follow the mouse pointer. The ball should bounce off the paddle.

For this initial version, give your project a special name (using “Save as”). For example, I am calling my copy as Bricks-1.
Feature Idea # 1: Bouncing ball

Get a “ball” sprite and make it bounce freely. After pressing SPACE BAR the ball should start bouncing around in the vertical direction.

Design:

I think this is so easy you can straightaway write the script. You already know how to get free motion in the horizontal direction. So, the only tricky challenge is to make the ball move up and down.

Hint: Think about setting the ball’s direction before it starts moving.

Feature Idea # 2: Paddle

Add the paddle. This involves two steps.

Step 1: The paddle should move left-right only and follow the mouse pointer.

Design:

How will you ensure that the paddle only moves horizontally?

Well, we can do that by keeping its Y coordinate fixed. So, whatever commands we use the paddle’s Y coordinate must not change.

And, how will you make the paddle follow the mouse pointer?

Since Y is not to change, the paddle’s X coordinate will continuously vary according to the pointer’s X coordinate. In other words, the paddle’s X coordinate will always be equal to the pointer’s X coordinate.

Hint: The “set x” command sets the sprite’s X coordinate, and the “mouse x” property (under “Sensing”) gives the pointer’s X coordinate.

Step 2: Make the ball bounce off the paddle.

Design:

First, you will need to teach the ball to sense when it touches the paddle.

You can use the “touching” condition in an IF statement.

“Bouncing” actually is a complex idea (think about reflection of light), but for now we will keep it simple: we will assume that bouncing essentially means turning around and moving away. The turning angle must be large. You can experiment and try different values.
Save Program Version 1

Before continuing to the next set of ideas, we will save our project. This way, we have a backup of our project that we can go back to if required for any reason.

Compare your program with my program at the link below.

**Bricks-1**: includes ideas 1 and 2 explained above.

Link: https://scratch.mit.edu/projects/104538051/

How to play the game:

1. Click on the “Green flag”: everything is reset to the original state.
2. Press SPACE BAR to start.
3. The ball starts bouncing up and down, and you can move the paddle left-right by moving the mouse pointer.

Next Set of Features/ideas:

1. Add bricks and make them work as expected. We will do this in three steps.
   a. Insert one brick sprite. The brick should appear at the start of the game and disappear when the ball touches it. Add a “score” variable, which will increase by 1 when the brick is hit by the ball.
   b. Next, duplicate the brick sprite to have multiple bricks. The script for each will be identical.
   c. Stop the game when all bricks have been hit, and declare victory.
2. Add a “speed” slider variable to control the speed of the ball.

For this version, make a copy of your project (using “Save as”) under a different name. For example, I am calling my copy as Bricks-2.

Feature Idea # 3: Bricks

*Add bricks and make them work as expected.*

Note that we can design just one brick sprite and then duplicate it to have as many as desired – I have 18 in my program. We will do this in three steps.

**Step 1**: Insert one brick sprite. The brick should appear at the start of the game and disappear when the ball touches it. Keep “score”, which will increase by 1 when the brick is hit by the ball.

**Design**:

Sensing that the ball has touched the brick is straightforward: it would be similar to
the way we did the paddle and ball above.

Score will be maintained by a variable called “score”. The brick needs to follow the following algorithm after the game starts:

   Show
   Wait (Do nothing) until it touches the ball
   Bump up the “score” variable by 1
   Hide

**Step 2:** Next, duplicate the brick sprite to have multiple bricks. The script for each will be identical.

This is self-explanatory! Just make sure the bricks are laid out nicely in rows.

**Step 3:** Stop the game when all bricks have been hit, and declare victory.

**Design:**

We now have the “score” variable to count the number of hits. When it equals the number of bricks, we will know that the game has been won. The stage can do this work using an additional script. Its algorithm will be as follows:

   Wait until the score is 18
   Declare “win”
   Stop the game

**Feature Idea # 4: Control speed**

*Allow the user to set the speed of the ball.*

**Design:**

How will you arrange things so that the user can set the speed of the ball?

The speed of the ball is decided by the “move” command. Bigger the input of “move”, higher the speed would be. So, you could use a variable in place of move’s input. By displaying this variable as a “slider” you can allow the user to control its value.

**Save as Program Version 2**

Before continuing to the next set of ideas, we will save our project. This way, we will have another backup of our project that we can go back to if required for any reason.

Compare your program with my program at the link below.
**Bricks-2:** includes ideas 3 and 4 explained above.

Link: https://scratch.mit.edu/projects/104712575/

**How to play the game:**

1. Click on the “Green flag”: everything is reset to the original state.
2. Set ball speed using the slider.
3. Press SPACE BAR to start the game.

**Final Set of Features/ideas:**

1. Implement the “number of lives” feature. We should see the number of lives as a set of balls. Every time the ball touches the ground (lower edge of the screen), we should see one ball less. When all lives are used, the program should declare that we lost.
2. Right now the ball doesn’t bounce off the bricks; it just goes through them. Make it bounce off the bricks.

For this final version, make a copy of your project (using “Save as”) under a different name. For example, I am calling my copy as Bricks-final.

**Feature Idea # 5: Number of lives**

Implement the “number of lives” feature. We should see the number of lives as number of balls. Every time the ball touches the ground (lower edge of the screen), we should see one ball less. When all lives are used, the program should declare that we lost.

*We will do this in 3 steps. In step 1, we will use a variable to count the number of lives. In step 2, we will actually show the number of lives (as balls) on the screen. And in step 3, we will ensure the game declares “You lose!” when all lives have been used.*

**Step 1:** Add a variable called “lives” which will track the number of “lives”. Every time the ball touches the ground, one life is lost. If the game allows, say 3 lives, the game should stop after 3 touches.

**Design:**

How will you make the ball sense that it has touched the bottom edge of the screen? Well, there is no special command “if touching bottom edge”, so we will have to improvise. We will color the bottom edge of the stage to represent “ground”, and then use the “touching color” condition in an IF statement. After every touch, we will decrement the lives variable. And, when lives becomes 0, we will stop the game.
Step 2: Show the “number of lives” on the screen.

**Design:**

We will do this by using a new sprite called “lives”. This sprite will have 3 costumes – showing 3, 2, and 1 ball(s) respectively. By tracking the “lives” variable, this sprite can show the appropriate costume. Its algorithm will be as below:

```plaintext
Show the 3 ball costume
Wait until lives = 2
Show the 2 ball costume
Wait until lives = 1
Show the 1 ball costume
Wait until lives = 0
Hide
```

Draw the sprite (along with its 3 costumes) and write a script to implement the above algorithm.

Step 3: Add another screen to declare when the game is lost.

**Design:**

The stage will have a new backdrop to declare “loss”. The ball sprite knows when the game is lost with the help of the following script:

![Script](image)

**Feature Idea # 6: Bounce off the bricks**

*Make the ball bounce off the bricks also.*

**Design:**

When the ball touches a brick, two things need to happen: (1) the ball should bounce off the brick, and (2) the brick should disappear. We have already done the second part by having the brick “sense” the touch. To do the first part, the ball will need to sense the touch as well.
If we ask both the ball and the brick to do the sensing, we risk having a “race condition” – a condition in which two entities are actively checking for a single event and one of them is likely to miss it. (See Concepts appendix for a detailed explanation of “race condition”).

To avoid this race condition, only one of the touching parties (ball or brick) should sense the touch and inform the other via broadcast.

**Save as Program Version “Final”**

Congratulations! You have completed all the main features of the game. As before, let’s save this project before continuing to the advanced ideas.

Compare your program with my program at the link below.

**Bricks-final**: includes ideas 5 and 6 explained above.

Link: [https://scratch.mit.edu/projects/104716301/](https://scratch.mit.edu/projects/104716301/)

**How to play the game:**

1. Click on the “Green flag”: everything is reset to the original state.
2. Set ball speed using the slider.
3. Press SPACE BAR to start the game.

**Advanced Features**

We can make several improvements in our program as listed below. These features are optional, so, implement only those that you find interesting and useful.

For this advanced version, make a copy of your project (using “Save as”) under a different name. For example, I am calling my copy as Bricks-adv.

**Feature Idea # 7: Correct bouncing**

*We have implemented the action of “bouncing” off the paddle (and bricks) quite arbitrarily. Make it more realistic (like the Scratch command “if on edge bounce”).*

**Design:**

Presently, our program just uses a “turn 150” command to implement bouncing off the paddle. The angle 150 is quite arbitrary. That is not how real-life bouncing works. The Scratch command “If on edge bounce” is a great example of realistic bouncing. So, let’s understand how that command works.

In Scratch, the “direction” property of a sprite indicates its angle with North. So, 0 means North, 90 means East, -90 means West, and so on.
When a sprite bounces off a vertical edge (left or right), its “direction” changes only in sign. So, 30 becomes -30, -110 becomes 110, and so on. When the sprite bounces off a horizontal edge (top or bottom), its “direction” after the bounce is 180-A where A is its direction before the bounce.

That is how the “If on edge bounce” command calculates the turning angle.

To keep things simple, we will only consider vertical bouncing for both the paddle and the bricks.

Modify your bouncing scripts and then compare with the solution given in the “Solutions” section.

**Feature Idea # 8: Layout of the bricks**

*Laying out the bricks manually is tedious and may not give a perfectly uniform look. Make this task of brick placement programmatic (i.e. through your scripts) and remove the manual error.***

**Design:**

The idea basically is to find a way to calculate the x and y coordinates of each brick and use the “Go To” command. We will use variables to do this.

Let’s say, variables $firstX$ and $firstY$ show the location of the top-left brick, and variables “$L$” and “$H$” show the length and height of each brick (plus some empty space). Using these variables, we can lay out the bricks neatly in two rows.

The following table shows a few examples of how these variables can be used to calculate positions of bricks:

<table>
<thead>
<tr>
<th>Brick</th>
<th>X coordinate</th>
<th>Y coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st brick in first row</td>
<td>$firstX$</td>
<td>$firstY$</td>
</tr>
<tr>
<td>5th brick in first row</td>
<td>$firstX + (4*L)$</td>
<td>$firstY$</td>
</tr>
<tr>
<td>3rd brick in second row</td>
<td>$firstX + (2*L)$</td>
<td>$firstY - (1*H)$</td>
</tr>
<tr>
<td>3rd brick in fourth row</td>
<td>$firstX + (2*L)$</td>
<td>$firstY - (3*H)$</td>
</tr>
</tbody>
</table>

In general, we can deduce the following equations to give the x and y coordinates of any brick:

$$x = firstX + (\text{column} \# - 1) \times L$$

$$y = firstY - (\text{row} \# - 1) \times H$$
For example, the following script will position a brick in the 4th place in the 1st row:

![Script example]

Someone in our Scratch program will need to set these four variables to appropriate values at the very beginning (when green flag is clicked). The stage would be a good candidate for that work. More importantly, the bricks must position themselves after the variables have been set. We can ensure this by using broadcasting. The stage will send a broadcast message after the variables have been set, and each brick will act upon receiving this message.

For example, the script below is for the 3rd brick in the 2nd row:

![Script example]

**Feature Idea # 9: Clones for bricks**

*You might have seen that it is tedious to use multiple sprites for bricks since any change requires a lot of duplication of work. If we use the idea of “clones”, we just need one brick sprite.*

**Design:**

The main (parent) brick will itself remain hidden, but create the required number of clones. It will also set x and y variables which each clone can use to position itself correctly.

**Algorithm:**

For the parent brick:
And then, each clone will run the following algorithm:

- Go to $x$, $y$
- Wait until touches ball
- Send message to ball
- Increment “score”
- Delete clone

**Save as Program Version “Advanced”**

Congratulations! You have completed all the advanced features of the game. As before, let's save this project.

Compare your program with my program at the link below.

**Bricks-adv**: includes the advanced features listed above.

Link: [https://scratch.mit.edu/projects/104716140/](https://scratch.mit.edu/projects/104716140/)

**How to play the game:**

1. Click on the “Green flag”: everything is reset to original state.
2. Set ball speed using the slider.
3. Press SPACE BAR to start the game.

**Additional Challenge(s)**

_Sometimes the ball starts moving perfectly horizontally. When that happens, there is no alternative but to restart the game. Can you add a script to detect this condition and alter the ball’s movement?_
**Design:**

We can use the “direction” property of the ball sprite to detect this condition. Fixing it is easy: just nudge the ball in a different direction by turning.

---

**Solutions to Feature Ideas**

**Feature Idea # 1:**

Script for the “Ball” sprite:

![Script for Ball](image)

*To ensure the ball starts its motion up and down.*

**Feature Idea # 2:**

**Step 1:**

Script for the “paddle” sprite:

![Script for Paddle](image)

*This property tracks the mouse pointer’s X position.*

**Feature Idea # 3:**

**Step 1:**

Script for the “brick” sprite:
Note that someone will need to initialize the “score” variable to 0 at the start of the game. You can really have any sprite do this. But, since this variable isn’t tied (or related) to any particular sprite, it’s a good policy to let the Stage do this initialization.

**Step 3:**

Script for the “stage”:

**Feature Idea # 4:**

The “speed” variable as a slider:
Script for the “ball” sprite:

Feature Idea # 5:

Step 1:

Script for the “ball” sprite:

* Without this wait statement, it is possible that the forever loop would sense a single touch multiple times and decrement the variable more than once. See it for yourself. With this “wait” added the other motion script would have taken the ball away preventing multiple decrements.
Step 2:
Script for the “lives” sprite:

- **when [ ] clicked**
- **switch costume to three**
- **show**
- **wait until lives = 2**
- **switch costume to two**
- **wait until lives = 1**
- **switch costume to one**
- **wait until lives = 0**
- **hide**

This will show 3 balls (i.e. 3 lives).
This indicates there are 2 lives left.
This indicates there is 1 life left.
This indicates there are no more lives left.

Step 3:
We can insert a broadcast in this script as shown below:

- **if lives = 0 then**
- **broadcast you lose and wait**
- **stop all**

The stage will receive the broadcast and use a new backdrop called “lose”:
Feature Idea #6:
Every brick sprite will send a broadcast after touching the ball:
![Script for brick broadcast](image1)

The ball sprite will receive the broadcast and bounce:
![Script for ball bounce](image2)

Feature Idea #7:
Script for the “ball” sprite:
![Script for ball movement](image3)

We will add a slight randomness to ensure the ball doesn’t get stuck in some fixed pattern.
![Script with randomness](image4)
**Feature Idea # 8:**
Solution is discussed in the design section itself.

**Feature Idea # 9:**
Refer to the program file for the advanced version.

**Additional challenge:**
Script for the “ball” sprite: