BUILDING MOBILE GAME SOLVERS

Build robots and develop algorithms which can automatically solve mobile games

Authors

Surya Penmetsa

Sudheesh Singanamalla

Building Mobile Game Solvers

Build Robots And Develop Algorithms Which Can Automatically Solve Mobile Games

SURYA PENMETSA[1]

SUDHEESH SINGANAMALLA[2]

April 15, 2016

[1] www.psurya.com
[2] www.sudheesh.info

.

Credits

This book is written by **Surya Penmetsa** and **Sudheesh Singanamalla** with help from the following amazing members of the Game Automators community.

- Piyush Kashyap
- Chandra S S Vamsi
- Nishi
- Karthik Shathiri
- Rajmani Arya
- Naveen Indala
- Piyush Agarwal

The book has been reviewed by:

- Nikhilendra Gudisa
- Sandeep Nadella
- Sharan Erukulla

We'd like to thank the following mighty people have assisted us in making all the work we do open source and free.

- Ramesh Akula
- Raja Poranki
- Priyanka Namburi
- Sathish Visanagiri

We would like to thank the Innovation Garage[1] and The Lakshya Foundation[2] for giving valuable suggestions all along and providing us with a great environment to work.

We'd also like to thank the other contributors: Aarti Barai, Sriram Kovela, Bharath Kumar, Vinay Kant, Aman Bhargava, Nisha Jacob, Satya Kesav, Anirudh Deshpande, Milind Sheth, Sreetam Das, Shashank Kasala, Milan Chatterjee, Chetan Rathore, Surendra Gurjar, Shubhi Saxena, Devendra Patil, Shivani Shukla, Spandan Pandey, Aroma Rodrigues, Rajat Soni and Priyansh Jain.

[1] https://www.facebook.com/TheInnovationGarage/
[2] http://www.thelakshyafoundation.org/

Contents

Preface

Making learning fun is extremely important. Many of us would have played video games when we were young, and what kept us glued and excited was the desire to make a high score. How awesome would it be to build your own robots and algorithms that play these games? How exciting would it be to watch your algorithm making it through the level that you found very hard?

This book is an attempt in that direction. It will introduce you on how to build systems and robots that can play mobile games. This approach will help you learn concepts of algorithms, electronics, image processing and machine learning; and have a lot of fun at the same time.

Background

I'm Surya Penmetsa, an ECE graduate from NIT Warangal class of 2015. I started working on such robots and algorithms during my final year at college. They could solve mobile games using concepts of electronics, image processing and machine learning. We used electronic sensors or image processing to sense what's on the screen; and then simulated touch physically or virtually at the appropriate locations.

I uploaded a video[3] of one of the robots that could play the game Piano Tiles on YouTube and it went viral. More than 200,000 people watched it. On popular demand, I also

[3] https://youtu.be/2TJ7itil1cc

uploaded a tutorial video[4] for the same, and it again got over 200,000 views. This shows the excitement and interest of people to solve games in an innovative manner.

I decided it was time for me to take the work to the next level. With the help of The Lakshya Foundation and Innovation Garage, I teamed up with students from NIT Warangal during the winter vacation[5] in December 2015, and Hackathon 5.0 in January 2016 to solve more games.

Most students had prior experience with electronic circuits, but very few knew image processing. I guided them through short lectures and provided them with online resources where they could learn further. They worked very hard, and grasped all required concepts very quickly. Together, we wrote algorithms for various games which beat human-level-performance.

We open-sourced all of the projects so that people around the world can replicate and build upon it. We also made video demonstrations for all projects and uploaded them on YouTube.

This book **Building Mobile Game Solvers** is a result of our work. I hope you learn a lot while reading the book and have a lot of fun.

We are open sourcing this book so that it can get better in quality by contributors across the world. We invite you, the reader, to be a contributor to the book to have more projects. The details on how to contribute are at the end of the book.

Bolsters Interest towards AI

Artificial Intelligence (AI) has been one of the fastest growing fields in the recent past. The book can also bolster the interest of the readers in this field. Once people learn building machines for game playing, they can expand into other areas in AI such as- natural language processing, robotics,

[4] https://youtu.be/8hlQ0MiowN8
[5] https://youtu.be/iDJW98c7uhg

computer vision, stock trading, medical diagnosis, etc.

Prerequisites

So what should you know in order to get started? This book has been carefully designed to help readers with or without experience in electronics and image processing. However, prior programming experience is recommended.

We covered concepts that are only relevant to solving mobile games in this book. In case you want to learn more, we have provided links at relevant places so that you can learn more.

Note

This book can be used to learn how fun electronics, image processing and machine learning are, but it cannot be independently used as a guide for any of these areas.

What this book covers

The book covers the following in each of the chapters.

Chapter 1, Introduction, covers the overview of each approach that we are going to use to solve the games in this book.

Chapter 2, Image Processing, explains what an image is actually made up of and how it can be analysed using different methods. It also teaches MATLAB commands that can be used for image processing.

Chapter 3, Electronics, introduces the sensors that can be used to sense what's on the screen and the various ways to simulate touch. It also talks about how a microcontroller can be used in to program the sensors and touch simulation.

Chapter 4, Solving Games, demonstrates how the concepts and methods of solving that we have learnt earlier can be used to solve specific games.

Conventions

In this book, you will find different formats of text that specify different types of information. For example, the paragraph text you are reading right now depicts normal text in the book. The bold in a bigger font size depicts the chapter names, headings or subheadings.

Codes are represented in blocks as follows-
Arduino code is represented this way.

```
void setup()
{
    // initialize serial communications at 9600 bps
    Serial.begin(9600);
}
```

Listing 1: Sample Arduino code

MATLAB code is represented this way.

```
% Reading an image
a = imread('input.jpg');
% Displaying the image
imshow(a);
```

Listing 2: Sample MATLAB code

Downloading the codes used in the book

All the codes that have been used in the book can be found at this link[6].

Downloading color images in the book

You can find the color images that have been used in the book at the following link[7].

[6] https://github.com/GameAutomators

[7] https://github.com/GameAutomators/eBook-Source/tree/master/Images

Reader feeback

We are always looking to improve the quality of the book. The feedback from the readers of our book is extremely important for us. Let us know what you think about the book by shooting an email to p.surya1994@gmail.com with the subject *Reader Feedback: Game Automators*. We promise to read each and every one of your emails.

If you find any mistake in the book text or the code- we would be grateful if you could report this on our github page[8].

[8] https://github.com/GameAutomators/eBook-Source/issues

1

Overview

The number of mobile devices has increased at a huge rate in the past decade. There are almost as many devices as there are people in the world. Phones have become an integral part of the lives of each one of us by helping us make phone calls, navigate using GPS, get high quality photos, play games, send text messages and many more. Further, with drastic improvements in the computational capability of the phones, the mobile gaming is a booming industry. In this chapter, we will learn different approaches to automate the mobile games.

Here are some of the games that we will learn to solve in this book: Find the Differences[1], Tic Tac Toe[2], Piano Tiles[3], Stick Hero[4], Flow Free[5] and Unblock Me[6]. Feel free to down-

[1] https://play.google.com/store/apps/details?id=com.ivanovandapps.ftdiaa3

[2] https://play.google.com/store/apps/details?id=com.pinkpointer.tictactoe

[3] https://play.google.com/store/apps/details?id=com.umonistudio.tile

[4] https://play.google.com/store/apps/details?id=com.ketchapp.stickhero

[5] https://play.google.com/store/apps/details?id=com.bigduckgames.flow

[6] https://play.google.com/store/apps/details?id=com.kiragames.unblockmefree

load these games and try them out on your phone.

We will learn three approaches in which we can automate the games. Do remember that these are just a few of the many approaches that you can use.

- Using Image Processing
- Using Electronics
- Using Image Processing and Electronics

Let us learn how to setup Android Debug Bridge (adb) tool which is used for communication between laptop and phone, before learning how each of the above mentioned approach works.

1.1 Setting up adb tool

Android Debug Bridge (adb) is a command line tool that allows the computer to communicate with attached Android devices. We will be using adb tool to capture screenshots of the mobile screen, simulate virtual touches and virtual swipes.

1.1.1 Setup

These are the steps that you have to follow to setup adb tool.

Step 1: Setting up adb tool

We will need a software called adb fastboot in order to be able to use the adb tool. Download the files in the provided link[7].

This will create a new file in your main hard drive (which is C: in most cases) named adb. We have to add this to

[7] http://forum.xda-developers.com/showthread.php?p=48915118

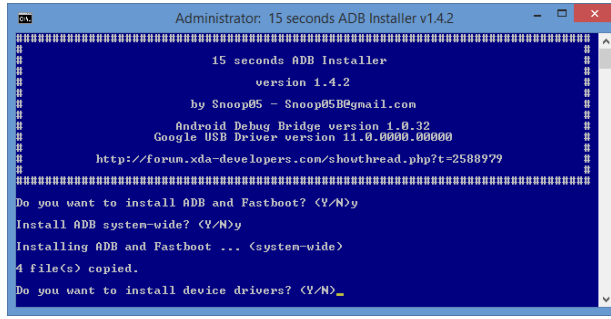


Figure 1.1: Screeshot while installing ADB

the environment path so that adb.exe can be used from anywhere in your computer.

Go to Control Panel -> System -> Advanced System Settings and a new window pop up. Go to the Advanced tab in this window and click Environment Variables. In the System Variables (the bottom section), choose the variable path and click edit to change it's value. At the end of the value, add the following. This will add the adb folder to your path that can be accessed from anywhere.

```
;C:\adb\
```

Step 2: Installing device drivers

You can find the drivers from Universal drivers[8] or Android developers website[9].

Open the Device Manager (click Start, type Device Manager, and press Enter), locate your device, right-click it and select Properties. You may see a yellow exclamation mark next to the device if its driver isn't installed properly. You

[8] http://adbdriver.com/downloads/

[9] http://developer.android.com/tools/extras/oem-usb.html#InstallingDriver

might also have to force windows to use the installed drivers for your phone.

Step 3: Enable USB debugging

To use adb with your Android device, you have to enable USB debugging.

You will find this in the developer options menu inside settings. If the developer options menu is not available, go into about phone menu in settings and tap the build number seven times to enable developer options menu. Also, make sure that you accept RSA fingerprint message shown in the device when its connected for the first time.

Step 4: Testing

To test whether adb is working properly, connect your Android device to your computer using USB cable and run the following command in the command prompt. This should list the devices.

```
adb devices
```

Also run the following command on MATLAB

```
system('adb devices')
```

The output of both the commands should look similar to this if your Android device is connected properly and detected.

```
List of devices attached
T038509BHC   device
```

If it says, command not found, it means adb tool is not setup properly, check if you complete step 1 correctly. If

your device is connected but nothing appears in the list, check if your device drivers are installed properly.

It is also possible to use adb tool over WiFi[10].

1.1.2 Basic Commands

Here are some commands that can be used with adb tool.

To take a screenshot of the device connected and store it on the sd card with the name 'screen.png'.

```
adb shell screencap -p /sdcard/screen.png
```

To copy the image stored previously on the sdcard on your computer's working directory.

```
1 adb pull /sdcard/screen.png
```

To tap on the screen at coordinates (x,y).

```
1 adb shell input tap x y
```

To swipe the screen from coordinates (x1, y1) to coordinates (x2,y2) with a delay of w milliseconds.

```
1 adb shell input swipe x1 y1 x2 y2 w
```

To remove screenshot that has been stored.

```
1 adb shell rm -f /sdcard/test.txt
```

That's all you need to know about adb tool. Next, let us cover each of the approaches in detail.

Remember, adb tool can only be used for Android phones. Instruments can be used to accomplish the same tasks that adb tool does in iOS.

[10] http://developer.android.com/tools/help/adb.html#wireless

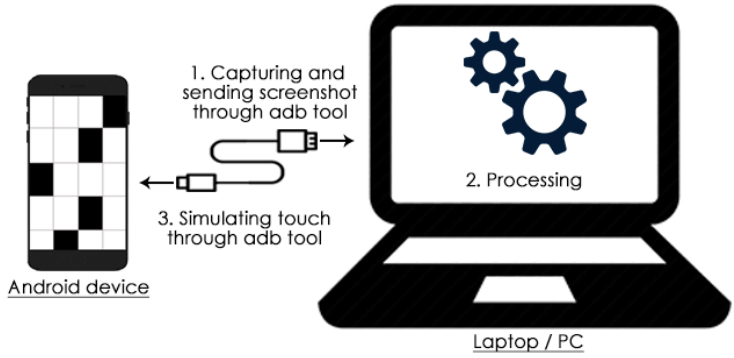


Figure 1.2: The image depicts the block diagram of the approach using adb tool and image processing for solving the games

1.2 Using Image Processing

In this approach, we use image processing and adb tool to automate the games. This has been illustrated in Fig. 1.2.

1.2.1 Approach

We use adb tool to take a screenshot of the phone screen and send over to the computer. Next, we use a set of image processing techniques to extract relevant features in the image. Depending on the features, we decide the appropriate action (touch or swipe) that must be taken and do that virtually using adb commands.

We can run the above steps in a loop to automate a game that needs repetition or has multiple levels.

1.2.2 Advantages of this approach

- We can get direct screenshots of the phone screen; hence the pixel values are reflected perfectly. So no preprocessing is required for the image.

- Since we can simulate precise touches and swipes.
- Complex algorithms can be implemented easily in case you are using MATLAB.
- Everything happening on the screen is visible to us unlike in the case of electronic circuits where the screen could be covered with the sensors or touch simulation circuitry.

1.2.3 Disadvantages of this approach

- The transfer of the screenshots to laptop, and simulating the touch takes about half a second. This is very slow if we are working on real time games that need quick response.

1.2.4 Games which can be solved

The Android games 'Stick Hero' and 'Find the Difference' can be solved using this approach because the games are not time bounded and tap on the screen is good enough to play these games.

1.3 Using Electronics

This is a way to automate the mobile games by using clever electronic circuitry placed on top of your phone. This has been illustrated in Fig. 1.3.

1.3.1 Approach

The microcontroller senses the inputs from a sensor that is used to detect what's on the screen, analyses the data using a logic that has been programmed into it and sends appropriate commands to the touch circuitry.

Refer to the *Electronics* chapter in the book to learn in detail about how the sensing and touching circuits work.

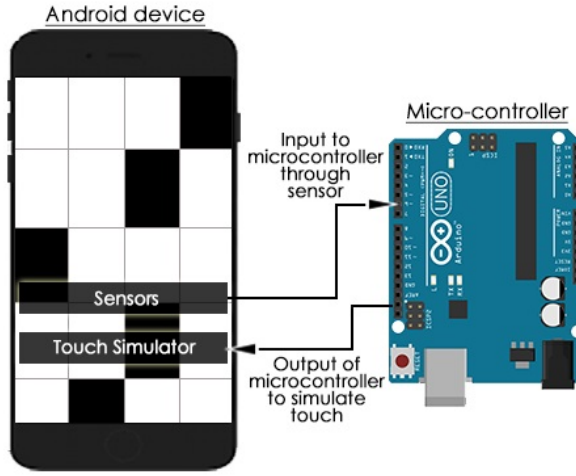


Figure 1.3: The image depicts the block diagram for a typical electronic circuit that can solve a game on the mobile

1.3.2 Advantages of this approach

- It's very fast unlike the previous approach. This speed is very important for solving many games.

1.3.3 Disadvantages of this approach

- External conditions such an ambient light might affect the working of the circuits.
- Setting up the touch part of the circuit takes some time.
- It's difficult to implement complex algorithms on Arduino.
- Circuit required to simulate swipe is complicated and involves mechanical parts.

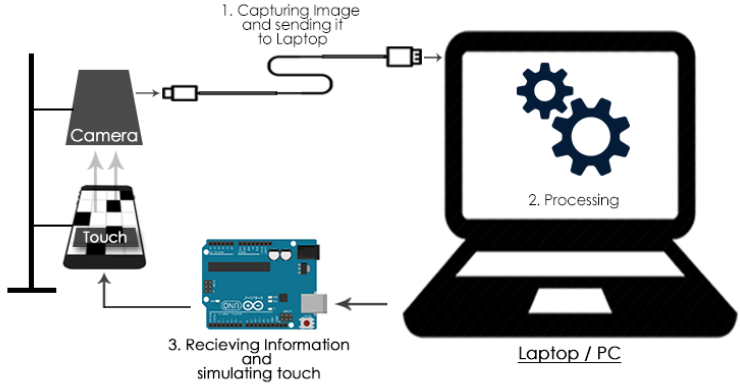


Figure 1.4: The image depicts the connection between the computer, microcontroller and the phone for game automation.

1.3.4 Games which can be solved

'Piano Tiles' and 'Ready Steady Bang' can be solved using this method because in these we just need to detect a intensity change on a small part of the screen using sensors and then simulate touch using electronic circuits.

1.4 Using Electronics and Image Processing

In this method we will be using the concepts of electronics and image processing together to automate the game. This helps us get over the problem we had which Approach 1 that it's slow. The concepts of image processing and electronics which we are going to use here is discussed in the following chapters.

1.4.1 Approach

We use a webcam which streams the video of the mobile screen into the computer that performs image processing

to extract relevant features, generates the duration of touch and sends it to the microcontroller which is used to simulate the touch on the phone screen using electronic circuitry. Recognize that we are not using adb tool here which makes this faster. This has been illustrated in Fig. 1.4.

For example, in the game 'Stick Hero', our webcam will capture the image of the screen and send it to the laptop where we can detect the pillars and the distace between them using image processing. Depending on the distance, we can send the information to the Arduino to simulate a touch for a specified time.

1.4.2 Advantages of this approach

- This method can be used to solve real time games like 'Flappy Bird'.
- This is faster than adb tool, even though not as fast as just using electornic circuits.

1.4.3 Disadvantages of this approach

- Complicated.
- Since external lighting influences the image captured by the webcam, we may have to change the algorithm accordingly each time.
- Setting the touch part of circuit becomes difficult.

1.4.4 Games which can be solved

The Android game 'Flappy Bird' can be solved using this approach because the games are real time bounded and quick tap on the screen is required to play these games.

We can increase the speed of image processing in this method by using libraries such as OpenCV or PIL instead of MATLAB's image processing toolbox. But let's stick to MATLAB in this book.

2

Image Processing

Image processing is rapidly growing field with a wide range of applications. Have you seen Facebook auto detect the faces of people in images you upload? That's image processing. Have you seen various filters that can be applied on an image in Instagram? That's image processing.

Image processing is also used in medical diagnosis, character recognition, robotic vision, emotion detection, etc., Image processing is a huge field and it's hard to cover everything in the book. So, in this chapter we cover the basics of image processing which is enough to automate various basic mobile games. We use MATLAB for implementing most of the image processing algorithms in this book but feel free to use any other software/package.

In this chapter, we will start by covering the basics of what an image is. Then, we show the various commands that are used in MATLAB and their outputs. We will also cover some important concepts of image processing used in game automation with the help of code.

2.1 What is an Image?

In this section, we learn what is an image and what does it consists of.

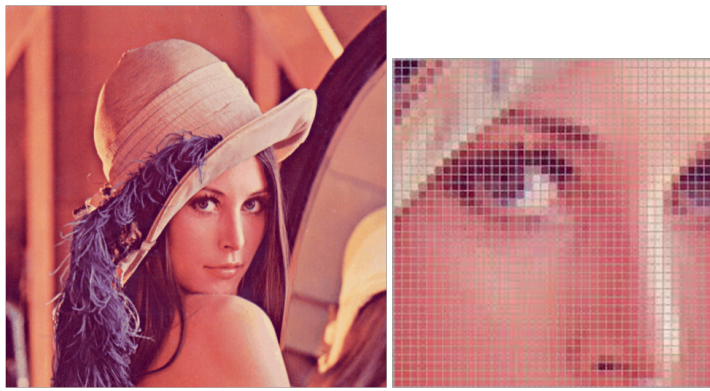


Figure 2.1: (a) The image of lena, (b) Image of lena zoomed in

2.1.1 Basics

Below is the image of Lena which we use often in image processing. New algorithms are experimented on this image as it consists of diverse chunks like- curly hair, plain background, human face, etc.,

When you zoom very closely into the image, you will start to realize that the image is made up of discrete squares as shown in Fig. 2.1(b). Each of the discrete square have their own color. These discrete sqares are called picture elements, or in short **pixels**.

Every image is created similarly by a two dimensional array of discrete square which have specific colors. These small discrete squares (or blocks) come together to form a bigger image. The image in Fig. 2.1(a) has a high resolution (512x512) and hence you are unable to see the discrete square with your eye directly. There are so many small blocks because of which our eye renders the image to be continous.

2.1.2 Image Resolution

The resolution of the image is the number of blocks in each of the directions in an image. It is represented by m x n when m is the number of pixels in x direction and n is the number of pixels in y direction.

For example, when I say the resolution of the image in Fig. 2.1(a) is 512 x 512, I mean that the number of blocks in x direction for the image is 512 and the number of blocks in y direction for the image is 512. That is a total of 262,144 pixels.

Let me give you another example, maybe a more intuitive one. When your phone manufacturer says that the resolution of the camera is 5 mega pixels (5 MP), what he means to say is that there will be a total of 5,000,000 pixels on the image that you take. So, the resolutions in x and y directions could be 2500 and 2000. So, the resolution of that image is 2500 x 2000.

2.2 Types of Images

The images are classified into three main categories which are mentioned below.

- Binary Image
- Grayscale Image
- RGB Image

Let us discuss each one of them now.

2.2.1 Binary Image

In a binary image, each of the pixels are either black or white. There is no other color as shown in Fig. 2.2.

Typically in a binary image, black is represented by the value 0 and white is represented by the value 1. This way, a binary image can be stored in a 2D matrix with just the numbers 0 and 1 in it. Fig.2.3 is an example of the same.

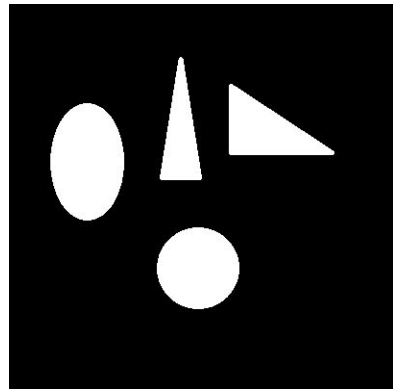


Figure 2.2: Binary Image

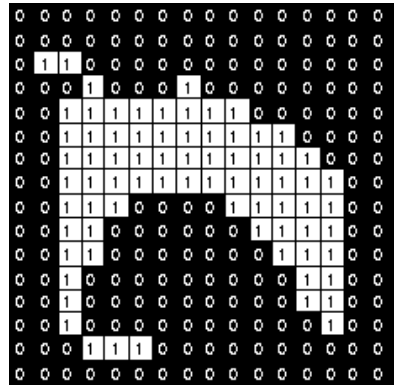


Figure 2.3: How Binary Image is stored

Figure 2.4: Grayscale image color values

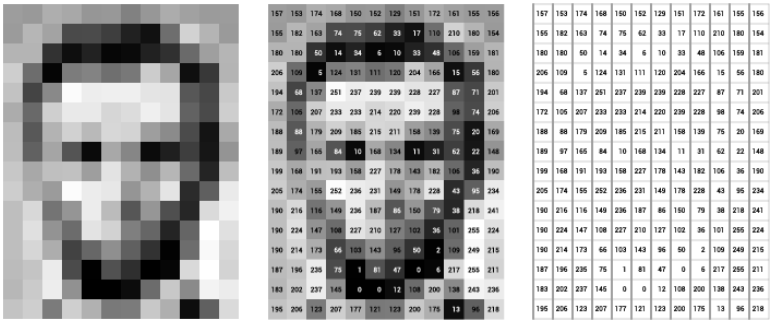


Figure 2.5: Grayscale Image

2.2.2 Grayscale Image

In a grayscale image, apart from having white and black; you can also have various shades of gray. For an 8-bit grayscale image, the value of each pixel varies from 0 to 255 where 0 represents pure black; 255 represents white; and all the values in between represent various shades of grey. The same is depicted in the Fig. 2.4 for clear understanding.

The grayscale image is stored in a 2D matrix with the values of each element varying between 0 and 255. An example of a grayscale image is shown Fig.2.5. The image of Lena in grayscale is depicted in Fig. 2.6.

2.2.3 RGB image

RGB stands for red, green and blue. Before getting into what an RGB image consists of, we have to understand how each and every color can be represented.

Figure 2.6: Grayscale Image of Lena

Representation of colors

Each of the pixels is represented by a single color and every color can be represented as a combination of three colors- red, green and blue. For example, white is the presence of all three colors: red, green and blue whereas black is the absense of these colors.

Fig.2.7. is the image of the popular color pallete in Microsoft Paint. You can select any color in the box in the right half of image. It's corresponding red, green and blue values are represented at the left bottom of the screen.

This is a 24-bit image, i.e., 8 bits in red, 8 bits in green and 8 bits in blue. Because it is 8 bits- the value of each color can vary between 0 and 255. 0 represents the absense of the color and 255 represents the presence. In this image, the color choosen has a value of red 255, green 128 and blue 64. It means the color has full red component, half green component and quarter blue component.

RGB Image Matrix

By using the above concept, any color can be represented as a combination of three colors- read, green and blue. Similarly, any image can be represented as combination of three

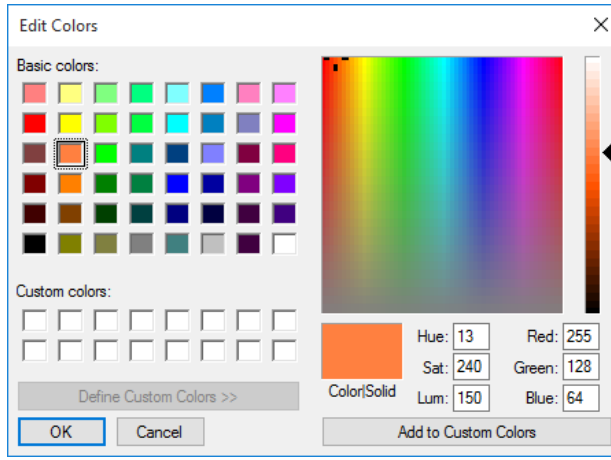


Figure 2.7: Color palette

layers. This is depicted in the Fih.2.8.

If these images are stored in a matirx, the size with be 200 x 150 x 3 where 3 represents the three layers.

2.3 Image Processing Techniques

We will use MATLAB for learning image processing but feel free to use other softwares if you have experience with them. MATLAB is very easy to learn and use but it's computationally intensive and slow. So, if for an application you need to process images faster, it's better to shift to C++ or Python and use image processing libraries such as OpenCV or PIL.

Now that you have understood the basics of what an image is, watch the first five videos "Image Processing in MATLAB" by The Motivated Engineer on YouTube[1].

Now, let's start learning various image processing techniques and see them in action with the help of MATLAB commands.

[1] https://www.youtube.com/playlist?list=PLmcMMZCV897oO5k7pfz23XkzXnCdcKbvn

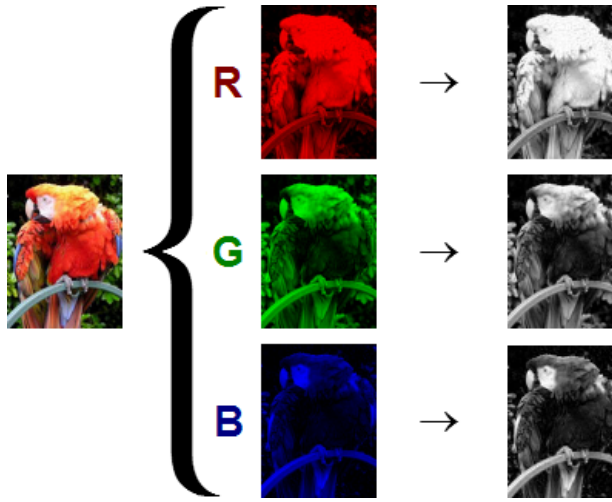


Figure 2.8: The three color layers of the image are shown on the right in grayscale form

2.3.1 Basic Commands

Let's start off by learning some basic commands in MATLAB to read, display and perform some basic operations on images such as crop, rotate and resize. To know more, or read detailed documentation for any of the commands, visit MATLAB help. You can use the following syntax for it.

```
doc command_name
```

imread

This command reads the image from a graphics file and stores it in a variable in the matrix form. The syntax for the imread operation is shown below.

```
% USAGE: varible_name = imread('file_location')
a = imread('input.jpg')
```

Figure 2.9: Image stored in input.png

The above command will store an image named 'input.png' into matrix form in the variable a. The image that we will be working with is shown in the Fig. 2.9.

The size of matrix a is decided by the dimensions of the image. Let us assume the size of 'object.png' be 266 x 400, then the matrix a will have 266 rows and 400 columns. If the object is an RGB image, the size of a will be 266 x 400 x 3 where 3 represents the layers of red, green and blue.

Make sure that the image 'object.png' is in your working directory for the code to work.

imshow

This command is used to show the image that has been stored in the matrix form.

```
% USAGE: imshow(varible_name)
imshow(a)
```

It will display the image stored in variable a in a new window as shown in Fig. 2.10.

For displaying the image in a new window, you can use

```
% USAGE: varible_name = imcrop(image, [x1 y1 l w])
figure, imshow(b)
```

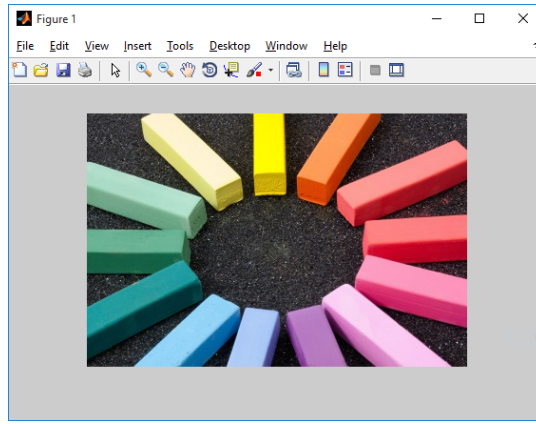


Figure 2.10: The window that pops up with imshow

where *figure* command create a new empty window where the image can be displayed. *imtool* is another function that you can use for displaying an image.

imcrop

This command crops the image according to the specified coordinates. The following syntax can be used to crop an image from the index values $(x1, y1)$ with the length l and width w.

```
b = imcrop(a, [90 90 200 300]); % cropping image
imshow(b) % displaying resulting image
```

It will crop the image a into a 151 x 151 x 3 image and store it in another variable b. You can use imshow to verify that the operation was performed correctly. The result is shown in Fig. 2.11.

imresize

This command resizes an image according to the specified scale, or to a specified size. Here's the syntax for using im-

Figure 2.11: Cropped image

Figure 2.12: Resized images

resize.

```
% USAGE: variable_name = imresize(image, scale)
c = imresize(a, 0.5); % resizing the image to half
figure, imshow(c)

% USAGE: variable_name = imresize(image, output_size)
d = imresize(a, [150 150]); % resizing image to give
    dimensions
figure, imshow(d)
```

The resulting images are shown in Fig. 2.12.

Figure 2.13: Image after rotation

imrotate

This command can be used to rotate the image by given angle (in degrees) in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for angle. *imrotate* makes the output image large enough to contain the entire rotated image. Here's the syntax for doing the same.

```
% USAGE: variable_name = imrotate(image, degrees)
e = imrotate(a, 15); % rotate 15 degree in clockwise
imshow(e)
```

By default, imrotate uses nearest neighbor interpolation, setting the values of pixels in output image that are outside the rotated image to 0 (zero). This is depicted in Fig. 2.13.

subplot

Creates axis in tiled positions. Whenever we need to display two or more images in one window, we use subplot.

```
subplot(m, n, p)
```

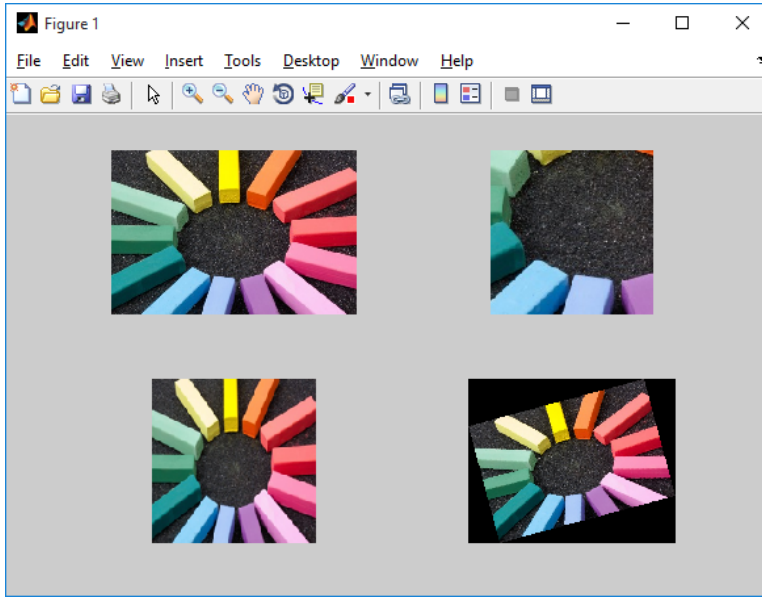


Figure 2.14: Output window after using subplot

It divides the current figure into an 'm x n' grid and creates an axes for a subplot in the position specified by p. **MATLAB** numbers its subplots by row, such that the first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If the axes already exists, then the *subplot(m,n,p)* makes the subplot in position p the current axes. Here's how it can be used.

```
subplot(2, 2, 1), imshow (a)
subplot(2, 2, 2), imshow (b)
subplot(2, 2, 3), imshow (c)
subplot(2, 2, 4), imshow (e)
```

subplot(2, 2, 1) will divide the figure into a 2 x 2 matrix. '1' is representing the position where the image a is going to be displayed in the 4 positions.

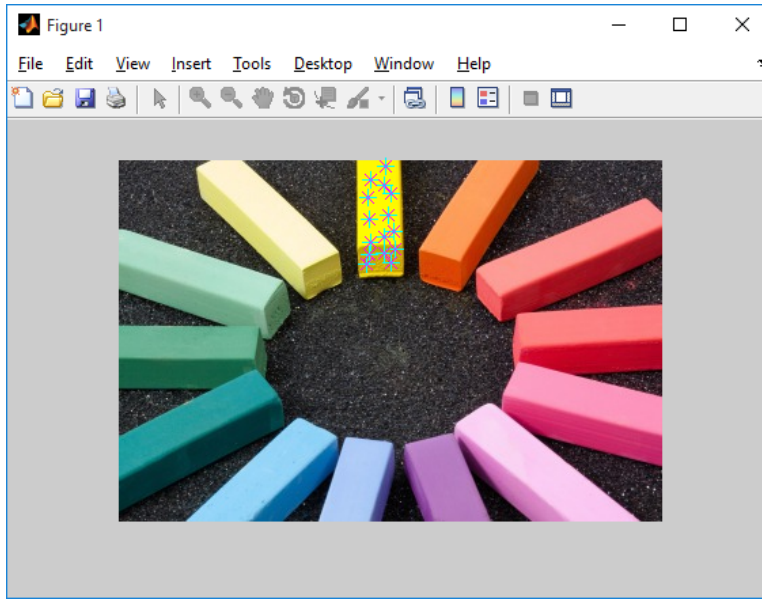


Figure 2.15: Choosing the points on window that pops up when you use impixel

2.3.2 Image Thresholding

Image thresholding is a simple way of detecting specific objects in the image. This technique converts RGB or grayscale images into binary image that has the object of interest seperated out. The binary image has the object of interest marked in white and rest of the image in black (or vice-versa) depending on how you threshold.

Choosing Threshold Values

The values for thresholding are choosen based upon the pixel intensities of the objects of interest in the image. There are multiple ways to look at these values.

One way is to use *imshow(img)* and use the pixel info tool to see the RGB values of specific pixels. You can also use *imtool(img)* and point the cursor at specific pixels to see the

RGB values at the bottom right of the window.

A better way to keep track of all these pixel values in an image is by using the function *impixel*. It is used with the following syntax.

```
%% Finding pixel values
values = impixel(img);
```

A new window pops up when the above command is executed where you can click on the object at a specific pixel and the RGB values of that pixel are stored in *values*. You can keep clicking at multiple point in the region of interest so that you have all the values stored in the variable *values*. You can stop choosing by double clicking on one of the pixels.

And there you have the pixel values that you need to threshold in the matrix *values*. When you finish selecting pixels, impixel returns a *m* x 3 matrix of RGB values in the supplied output argument. You can now look at the data inside *values* and see what are the ranges of red, green and blue intensity values in your image. Initialize the following variables depending on that data: *redMin*, *redMax*, *greenMin*, *greenMax*, *blueMin*, *blueMax*.

Here's a sample of the values.

```
%% Choosing the pixel values
redMin = 200; redMax = 255;
greenMin = 170; greenMax = 255;
blueMin = 0; blueMax = 15;
```

Splitting an image into RGB channels

An image can be split into RGB channels in MATLAB by using the following piece of code.

```
%% Splitting into channels
red = img(:,:,1); % filters first layer
green = img(:,:,2); % filters second layer
blue = img(:,:,3); % filters third layer
```

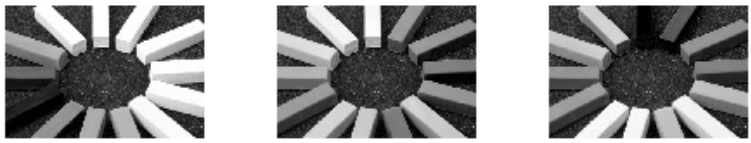


Figure 2.16: Displaying the RGB layers of input.png

```
% Displaying the result
figure, subplot(1,3,1), imshow(red)
subplot(1,3,2), imshow(green)
subplot(1,3,3), imshow(blue)
```

The output is shown below.

Thresholding in MATLAB

The required region or object must satisfy the condition that it's pixel values vary in the limits defined above. That can be implemented in one line in MATLAB using the following code.

```
%% Thresholding
out = red>=redMin & red<=redMax & green>=greenMin &
    green<=greenMax & blue>=blueMin & blue<=blueMax;
figure, imshow(out)
```

out will be a binary image with required object marked in white and others in black as shown in Fig. 2.17.

But understand that this code cannot be applied to all images. It can be applied only when the object to be detected is of the different color from the rest. In the image that we are using, direct thresholding was not good enough to detect the block perfectly. We how to do that using image enhancement in the next section.

Figure 2.17: Image after thresholding

2.3.3 Image Enhancement

The above image thresholding algorithm might not always work. Sometimes you will have to do some additional processing before you can detect the location of the object and that is what we will be learning now.

In this section, we will discuss the following morphological operations that you can perform on your obtained binary image to get a more satisfactory image.

- Dilation
- Erosion
- Filling holes
- Removing small objects

Let us learn about each of them now. Please note that the definitions provided here are over simplified for easier understanding. If you want to learn more in detail, take a course on Image Processing.

We will apply all these technique on the output we got from the previous section. Note that the following image is a zoomed version as it helps to see what's happening more clearly.

Figure 2.18: Close up of thresholded object

Figure 2.19: Image after dilation once

Dilation

Dilation is the process of expanding the shapes in the image.

It can also be used when the size of the shape in binary image is smaller than the actual object. Dilation can help make the shape size bigger. Here's the syntax for dilation of an image one time. The output is shown in Fig.2.19.

```
% Dilating the image once
out2 = bwmorph(out, 'dilate');
figure, imshow(out2)
```

You can also dilate an image multiple times by using an additional parameter as shown below. The output is shown

Figure 2.20: Image after dialtion multiple times

in Fig.2.20.

```
% Dilating the image 'n' times
n = 5;
out3 = bwmorph(out, 'dilate', n);
figure, imshow(out3)
```

The function *imdilate* can also be used for dilation. For learning more, type *doc imdilate* in your MATLAB command window.

Erosion

Erosion is the process of thining the object from the edges in the image. It can be used when the size of the binary image is larger than the actual object. Erosion can help make the object size smaller and filter out small object that have been unintentionally detected. Here's the syntax for erosion of an image one time. The output is shown in Fig.2.21.

```
% Eroding the image once
out2 = bwmorph(out, 'erode');
figure, imshow(out2)
```

Figure 2.21: Image after erosion once

Figure 2.22: Image after erosion multiple times

You can also erode an image multiple times by using an additional parameter as shown below. The output is shown in Fig.2.22.

```
% Eroding the image 'n' times
n = 5;
out3 = bwmorph(out, 'erode', n);
figure, imshow(out3)
```

Filling holes

Holes are black pixels in the image which are completely covered in all directions by white pixels. In other words,

Figure 2.23: Image after holes filled

hole is a set of pixels that cannot be reached by filling background from edge of image. The *imfill* function can be used to fill the holes. An example of how it can be used is shown below. The output is shown in Fig.2.23.

```
out2 = imfill(out, 'holes'); % Filling holes
figure, imshow(out2)
```

Removing small objects

Shapes in an image can be removed by using the following function. The area of the shape less than which have to be eliminated must be mentioned as one of the parameters for the function. The output is shown in Fig.2.24.

```
% remove object with area less than 100
area = 100;
out2 = bwareaopen(out, area);
figure, imshow(out2)
```

Once we have enhanced the image properly, we will have only the object(s) of interest left in the image. We will learn how to find the properties of these shapes in the next section.

Figure 2.24: Image after opening

2.3.4 Region Properties Extraction

In this section, we will learn how we can use the function 'regionprops' in MATLAB to detect various properties of objects that can be detected using steps in the previous sections. The input to the 'regionprops' function is a binary image, with the object marked in white and the background marked in black.

This command creates a *struct* variable in which it stores various properties for every region. The default properties are *Area*, *Centroid*, *BoundingBox*. The function also allows extracting a wide range of other properties that can be found in the MATLAB help or by typing in *doc regionprops* in the MATLAB command window.

Here's the syntax for using the function *regionprops*.

```
% Code to extract default properties of objects
Stats = regionprops(out2);

% Code to extract only centroids of objects
Stats = regionprops(out2, 'Centroid');
```

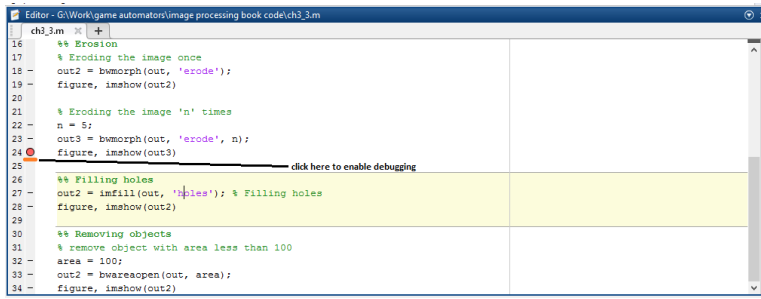


Figure 2.25: Image showing where you can click to add break points

2.3.5 Debugging in MATLAB

In this section, we will learn the basics of debugging and how we can debug in MATLAB.

What is Debugging?

Debugging is the process of stopping the execution of the code at a specific point so that you can analyse the state of the code there to find the mistakes. This is a very important feature of MATLAB that you have to understand and helps you save a lot of time while you have to find the mistake in your code. When the code stops the execution at a specific point, you can look at the values of all the variables there and see if everything's good.

Breakpoints

You can click on the margin left to the line numbers in MATLAB to create a breakpoint. The breakpoint is the point where you code stops executing.

You can see various features in MATLAB for debugging once you are at a break point.

Let's understand how each of these work now.

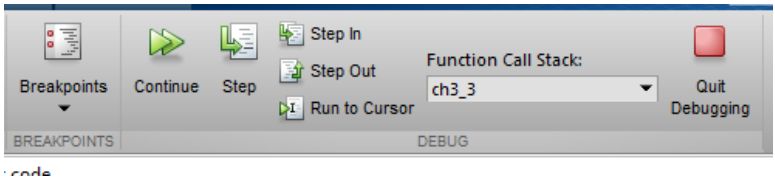


Figure 2.26: Debug control menu

Continue, as is name suggests, is used for continuing the execution of the code until the next breakpoint occurs.
Step takes you to the next step of the code.
Step in takes you into the function in the line that you are executing now.
Step out bring you out of a function that you may have got into using step in.

3

Electronics

In this chapter, we will cover the a brief introduction to electronics and an overview of the microcontroller Arduino and how it can be used to solve the mobile games. We will also learn various sensors that can used to detect what's on the screen and different ways of automating the touch.

We used Arduino in this book for the projects because it is easy to setup and use. Feel free to use the microcontroller that you are comfortable with.

3.1 Basic Electronics

Electronics deals with the use of circuits that involve various electrical components. Everything from the clock you see to find time to the smartphone in your hand connecting you to your friends every second are the applications of it. In this section, we will give a brief overview of the electronics that we need to get started with working on the book.

If you are a beginner, this video series on YouTube is a great way to start- **Building Electronic Circuits**[1]

It starts off by giving you an overview of the basics of the electronic components generally used and teaches you

[1] https://www.youtube.com/playlist?list=PLmcMMZCV897om7Wuqz882Jdp9lGj9HYHs

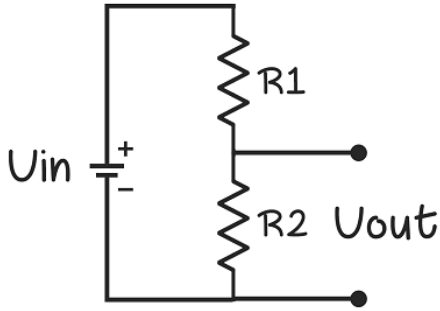


Figure 3.1: Voltage divider circuit

how to work on some cool projects such as infra red sensors and wearable circuits. Watching and practicing the circuits in the first six videos in this series would give you a decent level of understanding to continue reading the sections below. These videos are optional to watch for people who have worked on electronic circuits before.

3.1.1 Voltage Divider Circuit

A voltage divider circuit is commonly used in many electronic circuits for a wide range of applications including adjusting level of signal, measurement of voltages, etc. It is used in multimeter and wheatstone bridge. It produces an output voltage which is a fraction of the input voltage. The value of the output voltage depends on the values of the resistors used. We will use this circuit in a following section on sensors.

The output voltage of the above circuit can be calculated by the following equation. Read about ohm's law and work on some examples to know where we get the equation from.

$$V_{out} = \left(\frac{R2}{R1+R2}\right) V_{in}$$

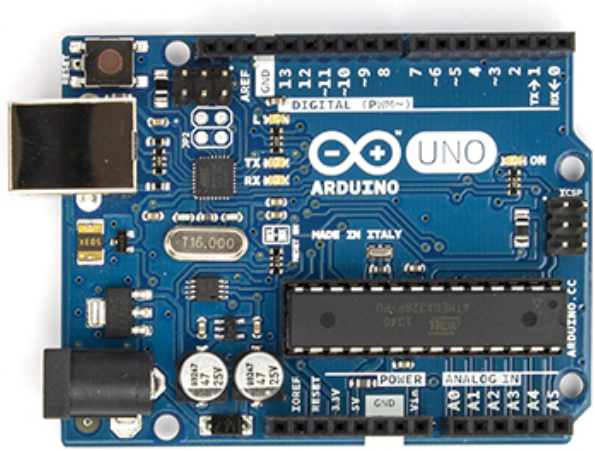


Figure 3.2:

Next, let us learn about how to use microcontrollers in building electronic circuits, specifically, Arduino.

3.2 Arduino

Arduino is an open-source, easy to use platform used for various electronics projects. It has a physical programmable circuit board. The capabilities of Arduino depends on the capabilities of the the microcontroller it's holding.

There are various types of Arduino's available in the market and the most popular one is called Arduino UNO. For instance, Arduino UNO uses ATMega328P and hence various specs depend on it.

In most applications, Arduino reads the inputs from the sensors, and takes action depending on the logic that has been programmed into it.

Jeremy Blum's Arduino tutorials[2] on the YouTube are

[2] https://www.youtube.com/playlist?list=PLV009FNOX7Tf-XSyghg2vrSYXw1QcCHaX

one of the best out there, watch the first three videos to get the basic understanding of Arduino.

3.2.1 Hardware

We will be talking mostly about Arduino UNO here. It has 20 I/O pins which means that these pins can be used to either take digital inputs, or provide digital outputs.

The pins are labelled as $D0$, $D1$, $D2$, ..., $D13$ and $A0$, $A1$, $A2$, ..., $A5$ which makes a total of 20 pins. The speciality of the pins labelled with A is that apart from digital input and output, they can also take analog inputs from sensors.

Values from various sensors can be read at the input pins and outputs can be activated accordingly.

3.2.2 Software

Arduino comes with a easy-to-use software that is available on their website[3]. It can be used to write code, compile it and then upload it onto the Arduino that can be used in your projects.

Here's how the code is organised in Arduino.

```
// initializations here
void setup(){
  // setup code here
}
void loop(){
  // loop code here
}
```

The *setup*() function only runs once and that is at the start of the program with the initializations above. The *loop*() runs continously after the setup is executed once. This code in the *loop*() function continues to run till the board is reset again.

[3] http://arduino.cc

Let us look at a sample code and try to understand how it works.

```
#define LED_PIN 13

void setup() {
    pinMode(LED_PIN, OUTPUT); // Enable pin 13 for
    digital output
}

void loop() {
    digitalWrite(LED_PIN, HIGH); // Turn on the LED
    delay(1000); // Wait one second (1000 millisec)
    digitalWrite(LED_PIN, LOW); // Turn off the LED
    delay(1000); // Wait one second
}
```

The *LED_PIN* variable is set to a value of 13. In the $setup()$ function, the pin D13 is set as a digital output pin by using the $pinMode$ function.

In $loop()$ function, the output state of the digital pin can be changed by setting it to high or low using the $digitalWrite$ function. A small time wait is executed by using the $delay()$ function. If an LED is connected to the output pin of the Arduino, this code will blink the LED with a time period of two seconds.

Next, let's understand how various sensors can be connected and used with Arduino.

3.3 Sensors for Automating Games

This section lists the various electronic sensors that can be used to detect the differences on the phone screen. The common and main idea for all of these sensors is that it senses the light and convert it into an understandable format. For example, LDR converts the change in light intensity that is falling on top of it into change in resistance.

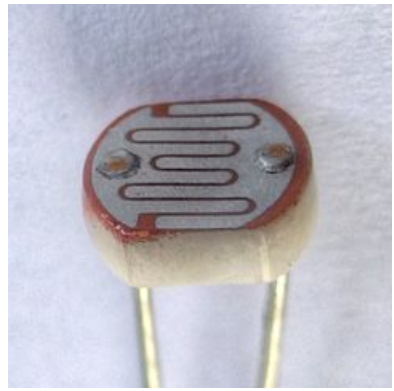


Figure 3.3: Light Dependent Resistor

3.3.1 Light Dependent Resistor(LDR)

An LDR is commonly used for wide range of applications because of it provides decently accurate information of the external lighting and at the same time economical. It is basically a light controlled resistor- which means that the resistance across its terminals changes according to the light incident on it. This can be used in projects where you want to sense the lighting in the surrondings. One of its common applications is to be used to turn on lights automatically in the evening. A video tutorial on how you can build such a circuit is here[4].

Working

It works on the principle of photo conductivity. When light is incident on top of a LDR, the electrons and holes are seperated-hence the conductivity increases i.e., resistivity decreases. When the light is not incident, they are very few freely moving holes and electrons- so the conductivity is less i.e., the resistivity is high.

[4] https://youtu.be/_uglvulpofQ

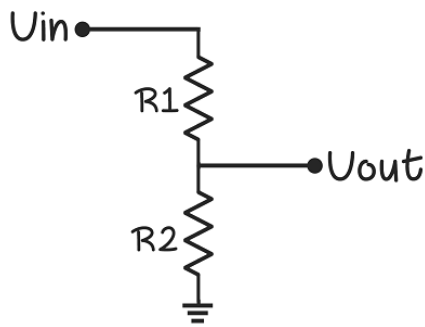


Figure 3.4: Voltage divider circuit

Circuit

Assume that LDR is connected at R1 and R2 is a resistor. As discussed in the previous chapter, the LDR can be used in a voltage divider circuit to convert this change in resistance because of the external light to change in voltage. We are doing this because the microcontroller can only detect the change in voltage.

This is how the LDR must be connected to the Arduino. This circuit diagram has been created using Fritzing[5].

Source code for using LDR with Arduino

This is the code that you can use in Arduino to read the values from the LDR with the appropriate circuit. We will display the values returned by the LDR on the serial monitor so that we can observe the changes in the value returned in real time.

Serial communication is used for data exchange between devices through the serial port. Let's setup it up here between the computer and Arduino.

```
void setup()
```

[5] http://fritzing.org/

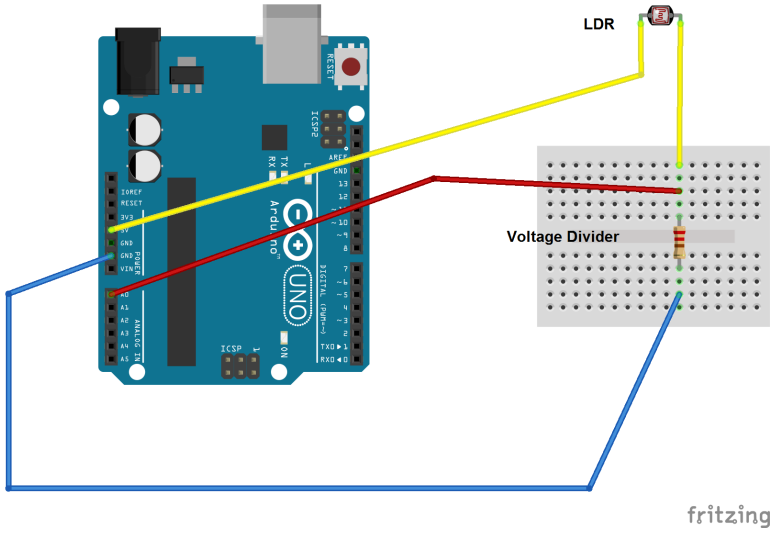


Figure 3.5: Circuit to connect LDR to Arduino

```
{
    // initialize serial communications at 9600 bps:
    Serial.begin(9600);
}
```

Let's connect the LDR input to A0 pin on Arduino.

```
void loop()
{
    // reading the value from sensor and storing it
    in a variable
    sensorValue = analogRead(A0);

    // print the output on serial monitor
    Serial.print("Sensor Value = ");
    Serial.println(sensorValue);

    // use a delay to see values clearly
    delay(20);
}
```

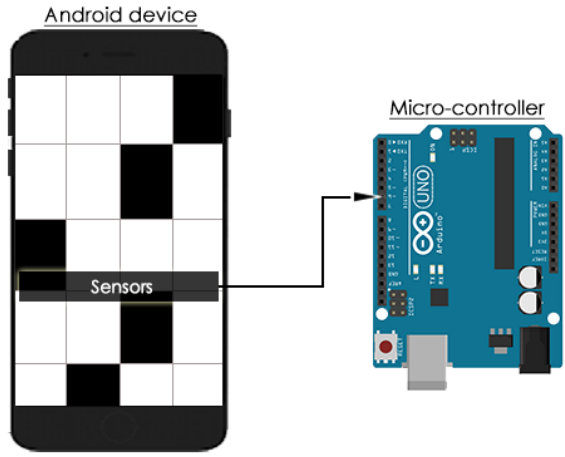


Figure 3.6: Connecting sensors to Arduino

Sensor placement on screen

Here's how the LDR can be placed on the top of the screen.

The output voltage from the LDR circuit changes depending on the screen color (either white or black). We have to choose a value in between both of them with we call the threshold to differentiate the colors. This threshold value can be found by watching the values of white and black on the serial monitor.

Once the sensor threshold value is found, we can use the following code to perform tasks accordingly.

```
void setup()
{
  thresholdValue = 500; // obtained from observation
    of serial monitor
}

void loop()
{
    // reading the value from sensor and storing it
    in a variable
    sensorValue = analogRead(A0);
```

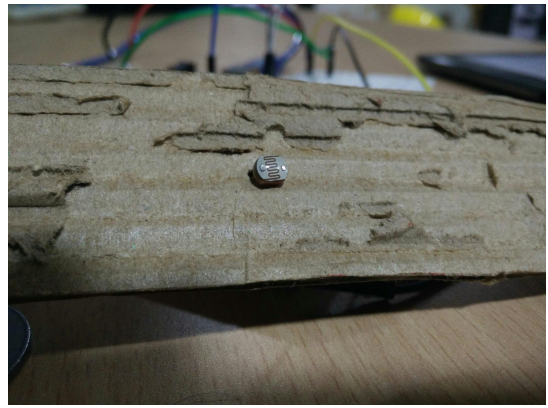


Figure 3.7: Shielding sensors

```
10
      if(sensorValue > thresholdValue)
12    {
        // perform task when screen white
14    }
      else
16    {
        // perform task when screen dark
18    }

20    delay(20);
}
```

Sensor protection from ambient light

This threshold value could change depending on the ambient light in the room. So, we use a cover (or shield) for the LDR to avoid that light to fall on the screen. Shielding is optional but we suggest you to do that. This will make the values that LDR returns consistent irrespective of the external light. We make sure that the color of the shield is dark so that it would block more light.

An LDR can be used to differentiate between bright and

dark regions on the screen. If you have a dark region, the LDR has high resistance and vice-versa. This is appropriately reflected in the values sensed by the microcontroller so that appropriate action can be taken.

This is used in the games Piano Tiles and Ready Steady Bang because the primary concept in this games is the identify the difference in intensities of light on screen.

3.3.2 RGB Sensors

Instead of an LDR, we can also use an RGB sensor to differentiate colors on the screen. The additional advantage of the RGB sensor from the LDR is that, it can know the exact color on the screen unlike LDR with which you can find the brightness of the color coming from the screen.

An RGB sensor can clearly differentiate between blue and green, whereas the same would be hard for an LDR to do.

3.4 Touch Simulation

Simulating touch is one of the critical parts of any game. There are multiple ways in which we can simulate the touch.

3.4.1 Mechanical

The touch can be simulated by a mechanical system consisting of a stylus connected to servo motor. A typical servo motor is a rotatory actuator that allows precise control of angle. The stylus must be connected to the servo in such a way that when the servo rotates, the stylus hits the screen.

This is the code that you can use to control the touch in this case. Let us assume that the mechanical arrangement is adjusted in such a way that the angle when the stylus touches the screen is 0 degrees and that the stylus doesn't touch the screen for 30 degrees.

```
// import the servo motor library
#include <Servo.h>

Servo myservo;  // create servo object to control a
    servo

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9
    to the servo object
}

void loop()
{
  // below are the angles for touching
  // 0 - touch, 30 - no touch

  myservo.write(30); // simulate no touch
  delay(1000);

  myservo.write(0); // simulate touch
  delay(1000);
}
```

But the problem with this method is that it is relatively slow. It is constrained by the speed of servo movement. To overcome this problem, we can use one of the following two ways.

3.4.2 Using Relay

This and the next method can only work on capacitive touch screens. For this we have to understand how they work. The electrodes apply a low voltage to the conductive layer creating a uniform electrostatic field. When a finger hits the screen a tiny electrical charge is transferred to the finger to complete the circuit creating a voltage drop at that point on the screen. The location of this voltage drop is recorded by the controller. This is shown in Fig. 3.8[6].

[6]Credit: The Curious Engineer (published with permission)

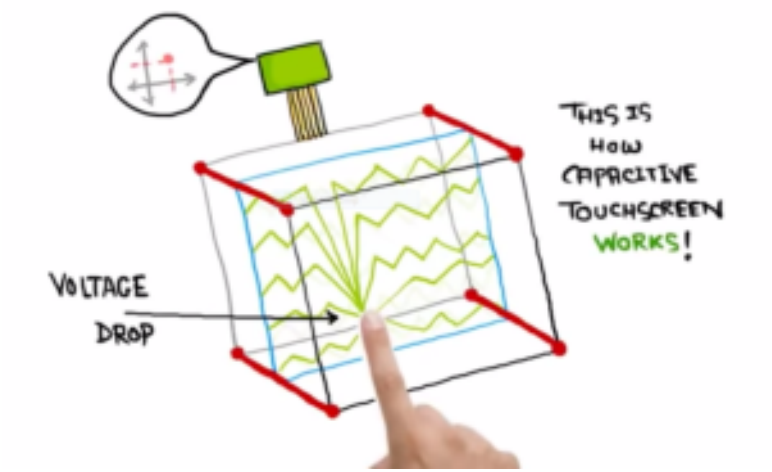


Figure 3.8: Depiction of how touch screen works

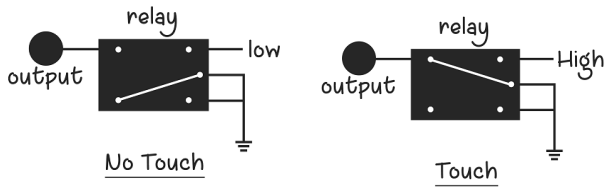


Figure 3.9: Relay internals

We are going to use this concept, except that in the place of a finger, we use the ground pin on the arduino to transfer the charge on the screen. To have more surface area on the display of the screen, we use a coin. Relays are directly connected to the output pin of the Arduino. As shown in the Fig. 3.9, it is equivalent to a touch if the voltage given is high as there is a path for the current to flow to the ground. It is equivalent to not touching, if the voltage given is low.

Fig.3.10 shows how you have to connect the relay in a circuit to simulate touch.

Here's the code to simulate touch and no touch alternatively with a time period of two seconds.

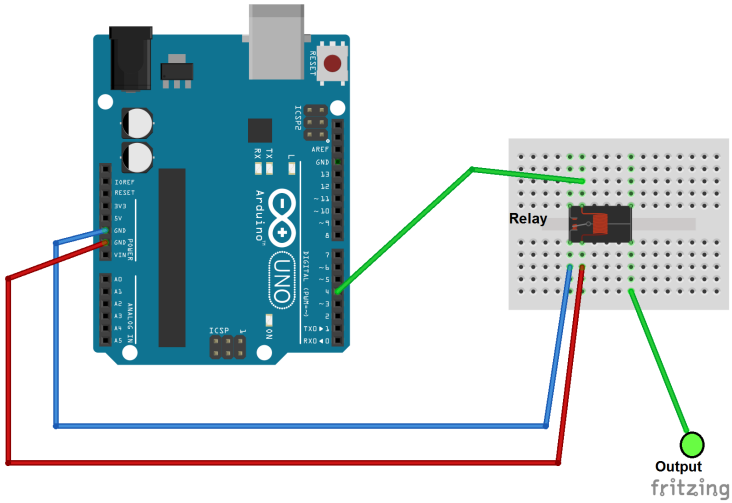


Figure 3.10: Connecting relay to Arduino

```
void setup()
{
  pinMode(4, OUTPUT);
}

void loop()
{
  // below are the values for touching

  digitalWrite(4, HIGH); // simulate no touch
  delay(1000);

  digitalWrite(4, LOW); // simulate touch
  delay(1000);
}
```

Touch simulation on screen

Here's how the touch circuit can be placed on the top of the screen.

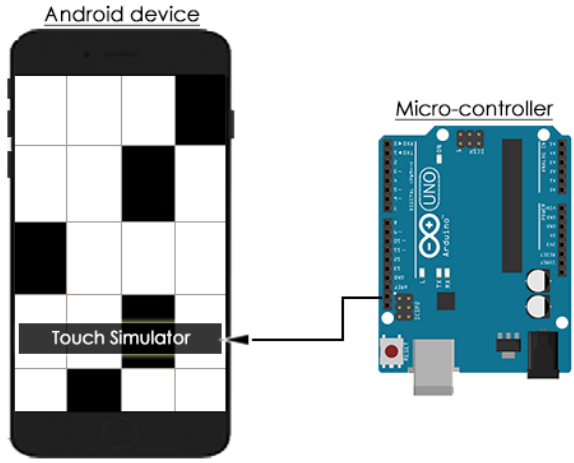


Figure 3.11: Touch simulation using Arduino

We should also make sure that there is enough contact between the wire from the output of the relay and the surface of the screen. Hence, we can use a coin or aluminium foil for this purpose.

Debugging

I must concede that getting the touch to work at first it hard. While using this circuit, try to have your phone in developer mode and tuen on the setting that says show touches and pointer location, so that you exactly understand when and where the screen is being touched.

If you circuit doesn't work directly, it's probably because the ground of your mobile and that of the circuit do not match. For overcoming this diffculty, you can try the following: make sure your circuit is not placed on an iron conductor, change the material used to increase the surface area of touch, try to plug both your circuit and the mobile into USB ports from the same computer, etc.,

3.4.3 Using Transistor

This method can work faster than a relay too. You have to use a transistor in place of a relay for switching between open circuit and ground. The code would be the same as above.

4

Solving Games

In this chapter, we cover specific examples of the concepts that we have learned earlier in this book. The chapter is split into sections depending on the approach that is used to solve the games. By the end of this chapter, you will be able to get enough insights into game automation so that you can start tackling new games. Note that the codes in this chapter might not work directly as you run them on your phone. The codes might have been written for a different phone screen resolution and you must make appropriate changes to the code before you run it.

4.1 3D Bowling

Game Description

In this game[1], the task of this player is to knock out as many pins as possible in a throw. You can throw the ball by swiping across the screen.

Difficulty Level: Easy

You can see a demo video of the working of this game at the following link[2].

[1] https://play.google.com/store/apps/details?id=com.threed.bowling

[2] https://youtu.be/fvfRw3w-E4s

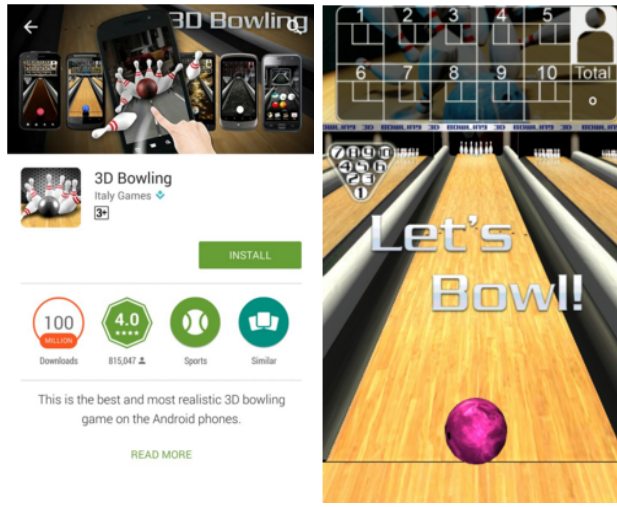
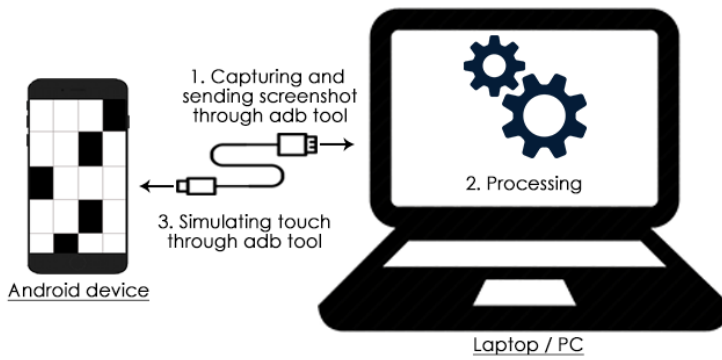


Figure 4.1: 3D bowling (a) on play store (b) gameplay

Overview

It's observed that a swipe across the center of the screen is the best way to drop maximum pins. That's what we are going to do with the code.

Block Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[3].

Step 1: Using ADB Tool to capture screenshot

The following command instantaneously takes the screenshot of the connected device and stores it in the SD card following the specified path.

```
system('adb shell screencap -p /sdcard/screen.png');
```

The following command pulls it from the SD card of the android device into the working system following the path specified.

```
system('adb pull /sdcard/screen.png');
```

The pulled image is stored in the form of a matrix of pixel values by the MATLAB.

Step 2: Choosing Points on the image

The pulled image is read and the two sets of coordinates are chosen such that they lie on the vertical axis that passes through the centre of the screen.

```
a = imread('screen.png');
imshow(a)
```

Step 3: Swipe across the screen

A swipe across the screen can be simulated by using the following command.

[3] https://github.com/GameAutomators/3D-Bowling

```
system('adb shell input swipe x1 y1 x2 y2');
```

where (x1,y1) and (x2,y2) are the cooridinates that were chosen.

Note: The swipe must be in the upward direction.

Step 4: Wait

We use a delay of 2 seconds for waiting for the animation of the swipe to be completed and ready for next throw. These two steps can be used in a loop to complete the whole game.

```
1 pause(2);
```

Conclusions

Ideally, the algorithm should be able to win each and every time because it's playing the best move every time. But there is a random element that has been programmed into the game because of which the ideal move doesn't always work.

This is an inefficient way to solve the game. A better method would be to choose the swipe direction depending on the location of the balls present on the screen.

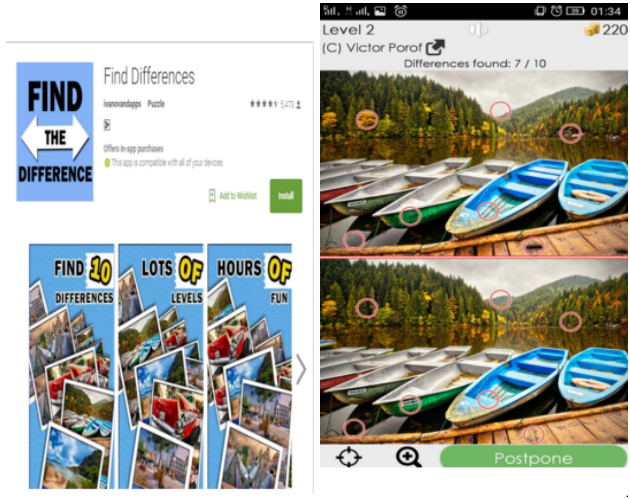


Figure 4.2: Find the difference (a) on play store (b) gameplay

4.2 Find the differences

Game Description

The game[4] has two images with ten minute differences. The aim of the player is to find the ten differences.
Difficulty Level: Easy

You can see a demo video of the working of this game at the following link[5].

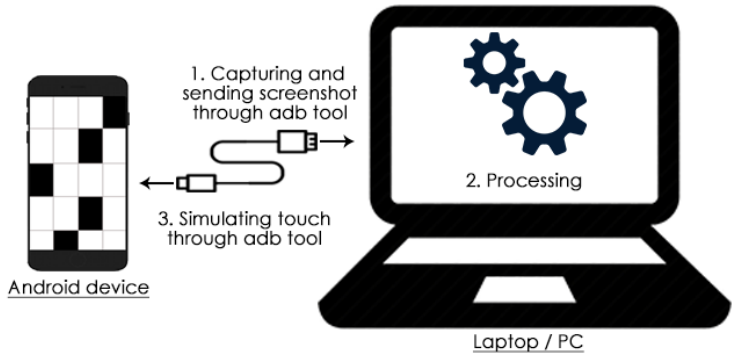
Overview

The image is divided into two parts and the differences are detected by using the processing the image. The differences are then made significant using an image enhancement and the final ten differences are obtained. Then ADB Tool library is used to simulate the touch on the found differences.

[4] https://play.google.com/store/apps/details?id=com.ivanovandapps.ftdiaa3

[5] https://youtu.be/vOTyJVKrqfk

Block Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[6].

Step 1: Using ADB Tool to capture screenshot

The following command instantaneously takes the screenshot of the connected device and stores it in the SD card following the specified path.

```
system('adb shell screencap -p /sdcard/screen.png');
```

The following command pulls it from the SD card of the android device into the working system following the path specified.

```
system('adb pull /sdcard/screen.png');
```

The pulled image is stored in the form of a matrix of pixel values by the MATLAB.

[6] https://github.com/GameAutomators/Find-The-Difference

Step 2: Image processing and enhancement

Both the images are separated and stored in two different matrices. The matrices are subtracted to get the difference matrix.

The difference matrix is converted into a binary image and the differences are made more significant by increasing the size of the differences by dilation. The centroids of the ten differences are found.

Step 3: Using ADB Tool to simulate touch

The following command taps at the point on the screen with the co-ordinates mentioned as (x, y). This is used to simulate touch at the centroid of the differences.

```
system('adb shell input tap x y');
```

where (x1,y1) and (x2,y2) are the cooridinates that were chosen.

Conclusions

This way, the computer solves the game very quickly, the speed at which humans can only dream off. This algorithm is robust and work for a few other find the difference games on the play store.

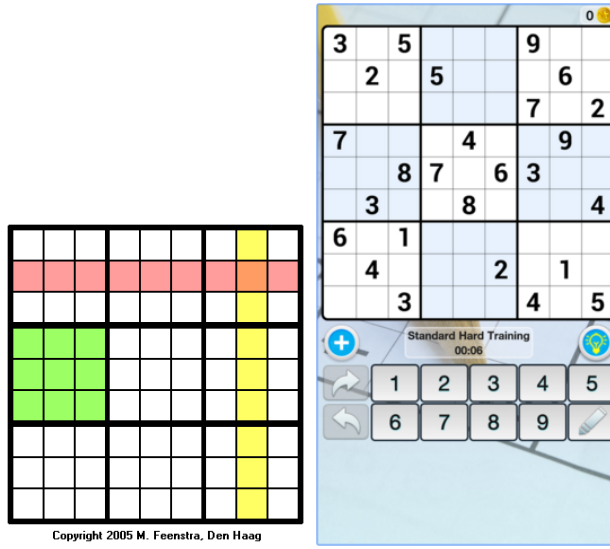


Figure 4.3: Sudoku (a) grid (b) gameplay

4.3 Sudoku

Game Description

This is a single player game[7]. The player has to solve sudoku mazes (9x9 grid). The objective of sudoku is to enter a digit from 1 through 9 in each cell, in such a way that:

- Each horizontal row (shown in pink) contains each digit exactly once.
- Each vertical column (shown in yellow) contains each digit exactly once.
- Each subgrid or region (shown in green) contains each digit exactly once.

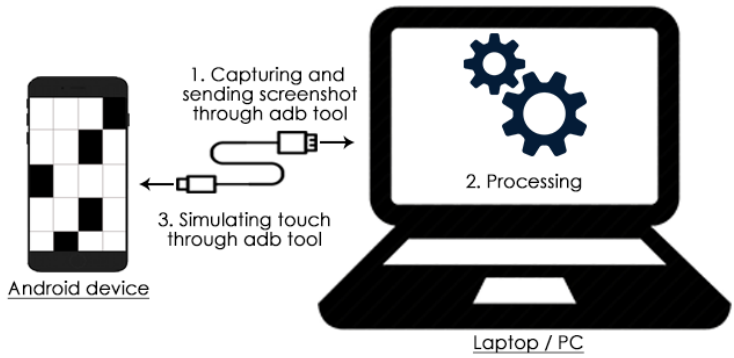
Difficulty Level: Moderate

[7] https://play.google.com/store/apps/details?id=le.lenovo.sudoku

Overview

First, using Image Processing all the numbers are recognized with their locations. Using algorithm, sudoku is solved and the numbers are maked in concerned box using adb tool.

Block Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[8].

Step 1: Using ADB Tool to capture screenshot

The following command instantaneously takes the screenshot of the connected device and stores it in the SD card following the specified path.

```
1 system('adb shell screencap -p /sdcard/screen.png');
```

The following command pulls it from the SD card of the android device into the working system following the path specified.

[8] https://github.com/GameAutomators/Sudoku-Game

```
system('adb pull /sdcard/screen.png');
```

The pulled image is stored in the form of a matrix of pixel values by the MATLAB.

Step 2: Image processing

Once the screenshot is obtained, smallest unit box is been croped out for recognization of number on it, using **OCR** (Optical Character Recognition). Recognized number is been stored in 9 x 9 matrix.

```
    a = rgb2gray(cimg);
    results = ocr(a ,'TextLayout','Block');
    rs2 = ocr(a,'TextLayout','Word');

    if(isempty(results.Words))
        A(i,j)=0;
    else
        A(i,j) = str2double(results.Words);
    end

    % bcoz ocr with block option is not detecting '8'

    if(8 == str2double(rs2.Words))
        A(i,j) = 8;
    end;
```

Step 3: Algorithm

The following command taps at the point on the screen with the co-ordinates mentioned as (x, y). This is used to simulate touch at the centroid of the differences.

```
  Find row, col of an unassigned cell
  If there is none, return true
  For digits from 1 to 9
    a) If there is no conflict for digit at row,col
```

```
        assign digit to row,col and recursively try
    fill in rest of grid
    b) If recursion successful, return true
    c) Else, remove digit and try another
  If all digits have been tried and nothing worked,
    return false
```

The algorithm checks all the possible ways using backtrack and tries to solve it under given rules and solved matrix is returned. The code is written in python. Tt takes command line string input of length 81 and return string output of same length using following commands.

```
% s is input string.

cmd = 'python sudoku.py ';
[status,out] = system([cmd s]);
```

Step 4: Using ADB Tool to simulate touch

The following command taps at the point on the screen with the co-ordinates mentioned as (x, y) and then tap on that specific number which is to be put there from bottom of screen. This is used to simulate touch at the appropriate points where we want to place the number.

```
system('adb shell input tap x y');
```

Algorithm Complexity

The Time complexity for sudoku solver written is $O(n^b)$ where n -> no. of possibilty of numbers for each cell i,e 9 and b -> no. of blank cell. The Space Complexity is Size of Sudoku and $O(n^b)$ for recursive call stack.

This can be seen be working backwards from only a single blank. If there is only one blank, then it have n possibilities that it must work through in the worst case. If there are two blanks, then it must work through n possibilities for the first blank and n possibilities for the second blank for each of the possibilities for the first blank. If there are three blanks, then you must work through n possibilities for the first blank. Each of those possibilities will yield a puzzle with two blanks that has n^2 possibilities.

This algorithm performs a depth-first search through the possible solutions. Each level of the graph represents the choices for a single square. The depth of the graph is the number of squares that need to be filled. With a branching factor of n and a depth of m, finding a solution in the graph has a worst-case performance of $O(n^m)$. So for hardest sudoku this can take minutes or hours also.

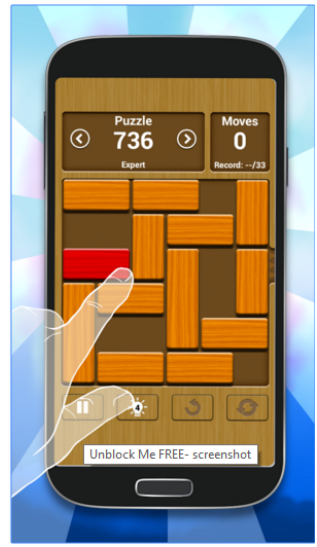


Figure 4.4: Unblockme gameplay

4.4 Unblock Me

Game Description

This is a single player game[9]. The goal is to unblock the red block out of the board by sliding the other blocks out of the way, unblock it with the minimal moves.
Difficulty Level: Moderate

You can see a demo video of the working of this game at this link[10].

Overview

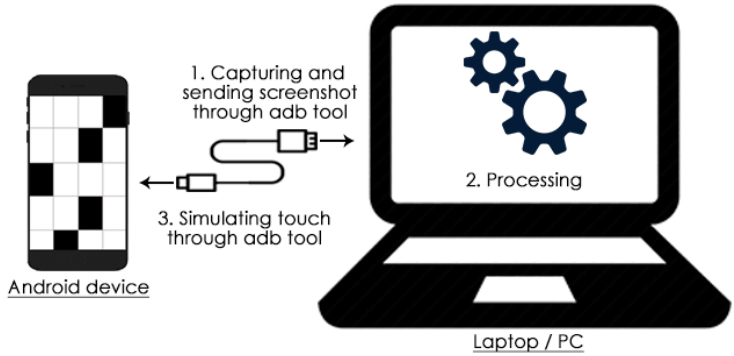
First, using image processing all the blocks alignment and position is detected using image processing in MATLAB. Using breadth first search algorithm, the game is solved and

[9] https://play.google.com/store/apps/details?id=com.kiragames.unblockmefree

[10] https://youtu.be/_-aNdgeLc5w

blocks are moved to free the red block. The swipes on the screen are simulated using adb tool.

Block Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[11].

Step 1: Using ADB Tool to capture screenshot

The following command instantaneously takes the screenshot of the connected device and stores it in the SD card following the specified path.

```
system('adb shell screencap -p /sdcard/screen.png');
```

The following command pulls it from the SD card of the android device into the working system following the path specified.

```
system('adb pull /sdcard/screen.png');
```

[11] https://github.com/GameAutomators/UnblockMe-Game

The pulled image is stored in the form of a matrix of pixel values by the MATLAB.

Step 2: Image processing

After taking screenshot main part of game is cropped out. Since Target block is of almost red color. So for specially that block, red color thresholding is performed.

```
ImBW = Im(:,:,2) < 10;
S = regionprops(ImBW,'BoundingBox','Area');
```

$S.BoundingBox$ gives the x, y, width and height of target block. Now to detect other blocks position other color thresholding is applied.

```
ImBW = Im(:, :, 1) > 220;
ImBW = imfill(ImBW,'holes');
S = regionprops(ImBW,'BoundingBox','Area');

for in=1:numel(S)
    if S(in).Area > 5000
        % take only those block whose Area is > 5000
    since regionprops may detect small on. of
    rectangles.
    end
end
```

Step 3: Algorithm

The algorithm uses a simple breadth first search to find a particular order of moves to free the red block.

```
Enqueue the current board
while Q not empty:
  Dequeue a board and examine it
  can block escape?
        he can! Ok
  he cant?
```

```
        for each possible board that can arise out of
    this one
            add board to END of Q
```

The code is written in C++. It takes command line string input and return string have order of moves. Using following commands in matlab we can run C++ program.

```
% s is input string.
cmd = 'unblock.exe';
[status,out] = system([cmd s]);
```

Step 4: Using ADB Tool to simulate touch

The following command swipes from the point (x1,y1) on the screen to the point (x2, y2).

```
system(['adb shell input swipe ' '' num2str(x1) ' '
    num2str(y1) ' ' num2str(x2) ' ' num2str(y2) ' 100
    ']);
```

Conclusions

This way, the game is automated. The code works for all levels in the game. You can run the code in a loop to solve all the 500 levels available in the game at once.

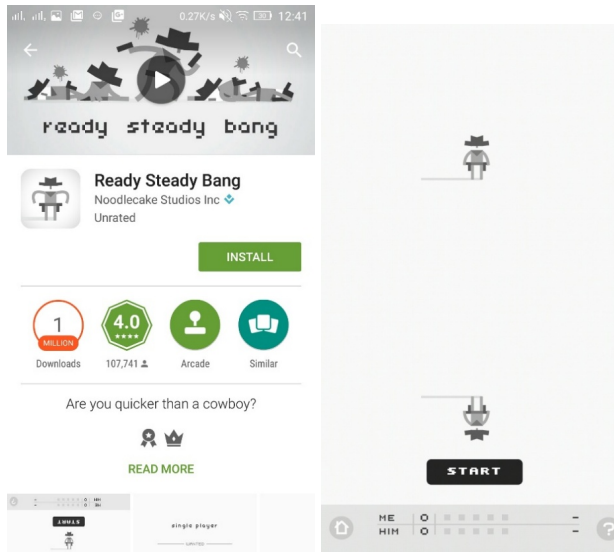


Figure 4.5: Bang (a) on playstore (b) gameplay

4.5 Bang

Game Description

The game[12] has two players, one is system and the other is player controlled. Both the players are allowed to shoot as soon as the word 'BANG' pops on the screen. The objective of the game is to fire the opponent before he does by tapping anywhere on the screen immediately after the 'BANG' appears.

Difficulty Level: Moderate

You can see a demo video of the working of this game at this link[13].

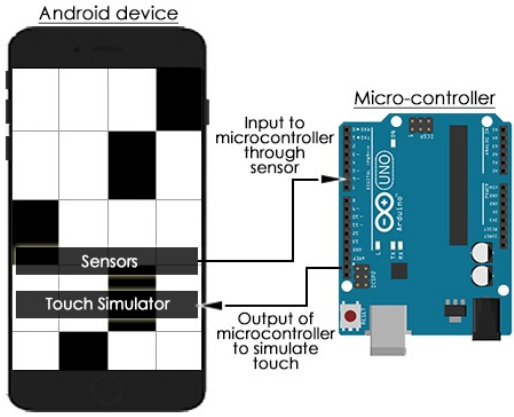
[12] https://play.google.com/store/apps/details?id=com.noodlecake.rsb

[13] https://youtu.be/riNjidXmOY4

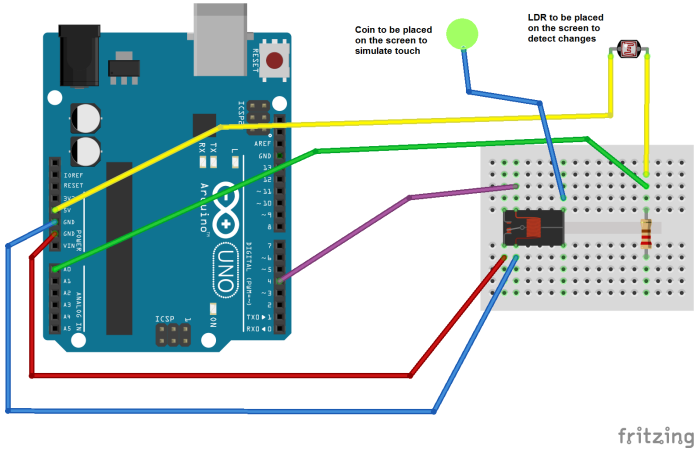
Overview

As soon as the word 'BANG' appears, the voltage output from the pin where the LDR is connected decreases. Whenever a drop in voltage is detected, the output pin is set to LOW and thus the Relay output is activated.

Block Diagram



Circuit Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[14].

Step 1: Sensor Placement

The LDR is fixed on the screen exactly where the 'BANG' appears. When the 'BANG' appears, the intensity of the light from the screen being detected by the LDR decreases.

Step 2: Touch Simulation

The coin is placed on the screen and the output from the relay is connected to it. When the relay output is grounded, it simulates a touch on the screen and when it is open circuited, it withdraws the touch.

Step 3: Arduino Code

The algorithm uses a simple breadth first search to find a particular order of moves to free the red block.

Initially, set the output pin HIGH.

```
digitalWrite(4,HIGH);
```

Read the input from the input pin, make a condition to check whether the input is less than the threshold voltage value (to be found experimentally) and check it with the condition.

```
int a=analogRead(A0);
if (a > Threshold value)
{
  :
  :
}
```

[14] https://github.com/GameAutomators/Bang

Whenever it satisfies the condition, simulate the touch.

Conclusions

This way, the the circuit plays the game for us. Because the reaction time of the circuit is very fast, it beats the computer easily.

Figure 4.6: Piano tiles gameplay

4.6 Piano Tiles

Game Description

The game[15] has tiles falling from the top of the screen. The player is expected to tap of the tiles (which are black in color) as quickly as possible without missing any.

Difficulty Level: Hard

You can see a demo video of the working of this game at this link[16].

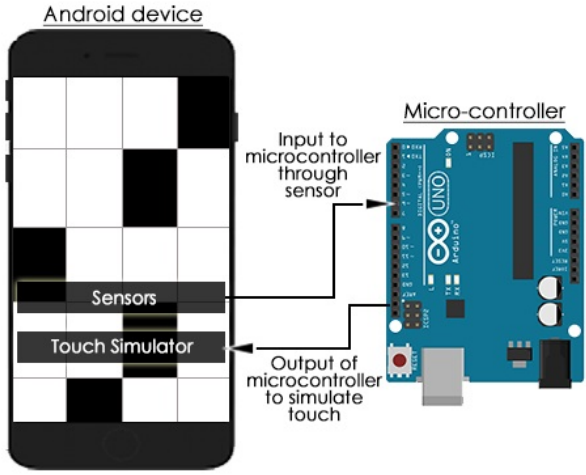
[15] https://play.google.com/store/apps/details?id=com.umonistudio.tile

[16] https://youtu.be/TQtS-OKW5Yo

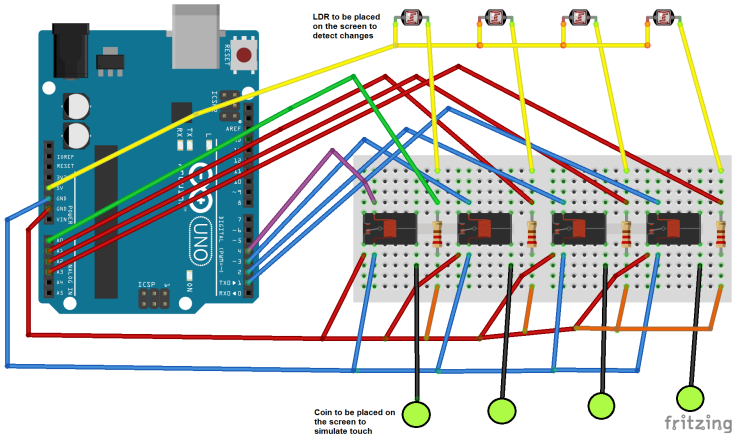
Overview

The color of the tile is sensed as black or white using LDR, and touch is simulated at appropriate locations on the screen using a logic programmed into the Arduino.

Block Diagram



Circuit Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[17].

Step 1: Sensor Placement

The LDRs are placed like a grid at the locations where the tiles will be falling through. When a black tile appears at the LDRs, the output voltage changes appropriately that can be observed on the Arduino.

Step 2: Touch Simulation

The coins are placed on the screen and the output from the relay is connected to it. When the relay output is grounded, it simulates a touch on the screen and when it is open circuited, it withdraws the touch. Appropriate coins are activated depending on the input from the LDRs.

Step 3: Arduino Code

Arduino reads the voltage drop across the LDR. Observe the voltage voltages for black and white tiles, choose a suitable threshold voltage say Vt. If voltage is less than the threshold voltage, then there is larger the drop across LDR, larger the resistance, which implies a Black tile and vice versa.

We have simulate touch accordingly. Tweak aroud with the paramters $delay1$ and $delay2$ until you get satisfactory results. The below code must be applied for each pair of sensors and actuators (total of 4).

```
int delay1 = 80;
int delay2 = 75;

if(analogRead(A5)<700)
{
digitalWrite(4, HIGH);
```

[17] https://github.com/GameAutomators/Piano-Tiles

```
  delay(delay2);
8 digitalWrite(4, LOW);
  delay(delay1);
10
  }
```

Conclusions

This way, you can build a circuit that can play the game Piano Tiles. This is a very interesting concept that can be applied over a wide range of other games.

4.7 Stick Hero

Game Description

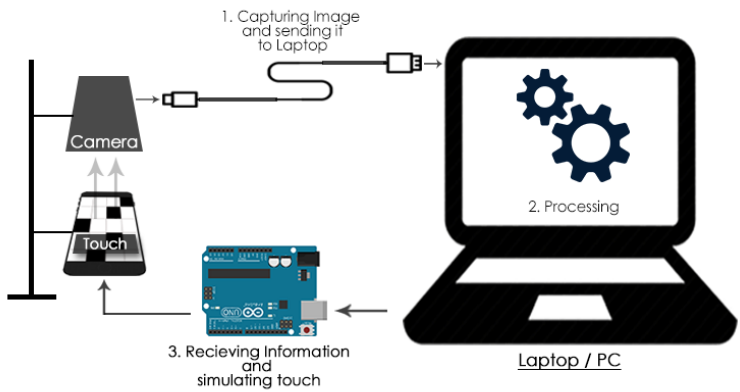
In this game[18], the player need to hold on the screen such that the stick that the man is holding increase its length such that the stick can be used to across the gap between two black pillars.
Difficulty Level: Hard

You can see a demo video of the working of this game at this link[19].

Overview

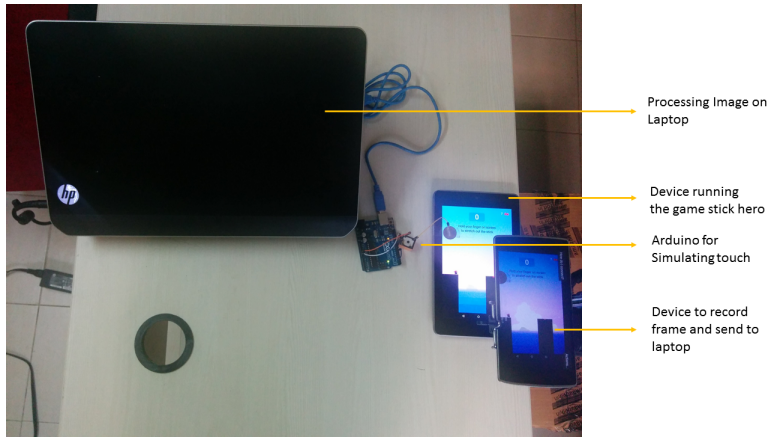
The black pillars are detected using image processing and the distance between them is calculated. This distance is converted to time by using a linear equation. The screen is touched by using the adb tool library.

Block Diagram



[18] https://play.google.com/store/apps/details?id=com.ketchapp.stickhero
[19] https://youtu.be/Na2GrGcEe9Q

Connection Diagram



Tutorial

Here's the step-wise tutorial to automate the game. The source code is avaiable here[20].

Step 1: Setup everything

Set everything up as shown in the connection diagram.

Step 2: Detecting the black pillars

This is done by using Matlab. The image that is captured from the IP Cam is processed and the locations of the black pillars are determined. The distance between the black pillars is calculated. and the corresponding data is sent to the Arduino through serial communication.

Step 3: Determining the duration of the touch

Based on the distance found in the above step, the corresponding data is sent to the Arduino through serial commu-

[20] https://github.com/GameAutomators/StickHero

nication. The duration of the touch is adjusted such that the stick exactly falls on the adjacent pillar.

Step 4: Simulating touch

Arduino reads the data from the serial port, and taps the screen for that specific interval of time.

```
    s = Serial.read();
    Serial.print(s); % reading data from serial port
    digitalWrite(7,HIGH);
    delay_x(s*3.28-0); // change multiplication
    factor depending on your phone
    digitalWrite(7,LOW);
```

Conclusions

This way, stick hero game can be automated. If setup properly, the system plays the game forever.

5

Contributing

This chapter details how you can contribute to the book.

5.1 Setting up to contribute

- Fork this repository GameAutomators/eBook-Source to your profile account
- `git clone https://github.com/<your github username>/eBook-Source`
- `git remote add upstream https://github.com/GameAutomators/eBook-Source`
- `git pull upstream master`

5.2 Making a contribution

Note: Never make a contribution from your master branch.

- Before starting to write, ensure that you've done `git pull upstream master` so that you're up to date with the main content.
- Checkout a new branch, this is like a copy of the content of the book so that you can make changes. `git`

`checkout -b BranchName` where `BranchName` can be anything depending on your contribution. For example, if you're writing an article on arduino you can do `git checkout -b MyArdunioDocument`

- Once the content is written. Do `git add <filename>` and `git push origin BranchName`
- Then headover to github and send a pull request.
- If you want to continue working on the same branch, you can do that or ideally switch back to master by doing `git checkout master`
- Pull back from `upstream` by doing step 1 before starting to make the next contribution to the book.

We look forward to your contributions to the book.