

Unit 1: Welcome to Visual Programming

Broadcast, Animations, and Music!

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 4: The student can use programming as a creative tool.
- 5: The student can describe the combination of abstractions used to represent data.
- 6: The student can explain how binary sequences are used to represent digital data.
- 7: The student can develop an abstraction.
- 9: The student can use models and simulations to raise and answer questions.
- 28: The student can analyze how computing affects communication, interaction, and cognition.
- 29: The student can connect computing with innovations in other fields.
- 30: The student can analyze the beneficial and harmful effects of computing.
- 31: The student can connect computing within economic, social, and cultural contexts.

Readings/Lectures

- Blown to Bits: Chapter 1 (<http://www.bitsbook.com/wp-content/uploads/2008/12/chapter1.pdf>)
- Reading 1.01: What is Abstraction? (/bjc-course/curriculum/01-welcome/readings/01-what-is-abstraction)
- Reading 1.02: More on Abstraction (/bjc-course/curriculum/01-welcome/readings/02-more-on-abstraction)
- Reading 1.03: Binary and Hexadecimal Numbers (/bjc-course/curriculum/01-welcome/readings/03-binary-and-hexadecimal)

External Resources

- Lecture Video: Binary Hex Decimal (<http://www.screencast.com/t/c2tp610y1tx6>)
- Why software is eating the world (<http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html>) (Wall Street Journal) (DEAD LINK)

Labs/Exercises

- Lab 1.01: Conversion Exercise (/bjc-course/curriculum/01-welcome/labs/01-conversion)
- Lab 1.02: Welcome to Visual Programming (/bjc-course/curriculum/01-welcome/labs/02-exploring-visual-programming)
- Lab 1.03: Lights, Camera, Action (/bjc-course/curriculum/01-welcome/labs/03-lights-camera-action)

Blown to Bits

Your Life, Liberty, and Happiness After the Digital Explosion

Hal Abelson
Ken Ledeen
Harry Lewis



Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/aw

Library of Congress Cataloging-in-Publication Data:

Abelson, Harold.

Blown to bits : your life, liberty, and happiness after the digital explosion / Hal Abelson, Ken Ledeen, Harry Lewis.

p. cm.

ISBN 0-13-713559-9 (hardback : alk. paper) 1. Computers and civilization. 2. Information technology—Technological innovations. 3. Digital media. I. Ledeen, Ken, 1946- II. Lewis, Harry R. III. Title.

QA76.9.C66A245 2008

303.48'33—dc22

2008005910

Copyright © 2008 Hal Abelson, Ken Ledeen, and Harry Lewis

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. To view a copy of this license visit
<http://creativecommons.org/licenses/by-nc-sa/3.0/us/> or send a letter to Creative Commons
171 Second Street, Suite 300, San Francisco, California, 94105, USA.

For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

ISBN-13: 978-0-13-713559-2

ISBN-10: 0-13-713559-9

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
Third printing December 2008

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to <http://www.informit.com/onlineedition>
- Complete the brief registration form
- Enter the coupon code 9SD6-IQLD-ZDNI-AGEC-AG6L

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

Editor in Chief

Mark Taub

Acquisitions Editor

Greg Doench

Development Editor

Michael Thurston

Managing Editor

Gina Kanouse

Senior Project Editor

Kristy Hart

Copy Editor

Water Crest Publishing, Inc.

Indexer

Erika Millen

Proofreader

Williams Woods Publishing Services

Publishing Coordinator

Michelle Housley

Interior Designer and Composition

Nonie Ratcliff

Cover Designer

Chuti Prasertsith

CHAPTER 1

Digital Explosion

Why Is It Happening, and What Is at Stake?

On September 19, 2007, while driving alone near Seattle on her way to work, Tanya Rider went off the road and crashed into a ravine.* For eight days, she was trapped upside down in the wreckage of her car. Severely dehydrated and suffering from injuries to her leg and shoulder, she nearly died of kidney failure. Fortunately, rescuers ultimately found her. She spent months recuperating in a medical facility. Happily, she was able to go home for Christmas.

Tanya's story is not just about a woman, an accident, and a rescue. It is a story about bits—the zeroes and ones that make up all our cell phone conversations, bank records, and everything else that gets communicated or stored using modern electronics.

Tanya was found because cell phone companies keep records of cell phone locations. When you carry your cell phone, it regularly sends out a digital “ping,” a few bits conveying a “Here I am!” message. Your phone keeps “pinging” as long as it remains turned on. Nearby cell phone towers pick up the pings and send them on to your cellular service provider. Your cell phone company uses the pings to direct your incoming calls to the right cell phone towers. Tanya's cell phone company, Verizon, still had a record of the last location of her cell phone, even after the phone had gone dead. That is how the police found her.

So why did it take more than a week?

If a woman disappears, her husband can't just make the police find her by tracing her cell phone records. She has a privacy right, and maybe she has good reason to leave town without telling her husband where she is going. In

* Citations of facts and sources appear at the end of the book. A page number and a phrase identify the passage.

Tanya's case, her bank account showed some activity (more bits!) after her disappearance, and the police could not classify her as a "missing person." In fact, that activity was by her husband. Through some misunderstanding, the police thought he did not have access to the account. Only when the police suspected Tanya's husband of involvement in her disappearance did they have legal access to the cell phone records. Had they continued to act on the true presumption that he was blameless, Tanya might never have been found.

New technologies interacted in an odd way with evolving standards of privacy, telecommunications, and criminal law. The explosive combination almost cost Tanya Rider her life. Her story is dramatic, but every day we encounter unexpected consequences of data flows that could not have happened a few years ago.

When you have finished reading this book, you should see the world in a different way. You should hear a story from a friend or on a newscast and say to yourself, "that's really a bits story," even if no one mentions anything digital. The movements of physical objects and the actions of flesh and blood human beings are only the surface. To understand what is really going on, you have to see the virtual world, the eerie flow of bits steering the events of life.

This book is your guide to this new world.

The Explosion of Bits, and Everything Else

The world changed very suddenly. Almost everything is stored in a computer somewhere. Court records, grocery purchases, precious family photos, pointless radio programs.... Computers contain a lot of stuff that isn't useful today but somebody thinks might someday come in handy. It is all being reduced to zeroes and ones—"bits." The bits are stashed on disks of home computers and in the data centers of big corporations and government agencies. The disks can hold so many bits that there is no need to pick and choose what gets remembered.

So much digital information, misinformation, data, and garbage is being squirreled away that most of it will be seen only by computers, never by human eyes. And computers are getting better and better at extracting meaning from all those bits—finding patterns that sometimes solve crimes and make useful suggestions, and sometimes reveal things about us we did not expect others to know.

The March 2008 resignation of Eliot Spitzer as Governor of New York is a bits story as well as a prostitution story. Under anti-money laundering (AML) rules, banks must report transactions of more than \$10,000 to federal regulators. None of Spitzer's alleged payments reached that threshold, but his

bank's computer found that transfers of smaller sums formed a suspicious pattern. The AML rules exist to fight terrorism and organized crime. But while the computer was monitoring small banking transactions in search of big-time crimes, it exposed a simple payment for services rendered that brought down the Governor.

Once something is on a computer, it can replicate and move around the world in a heartbeat. Making a million perfect copies takes but an instant—copies of things we want everyone in the world to see, and also copies of things that weren't meant to be copied at all.

The digital explosion is changing the world as much as printing once did—and some of the changes are catching us unaware, blowing to bits our assumptions about the way the world works.

When we observe the digital explosion at all, it can seem benign, amusing, or even utopian. Instead of sending prints through the mail to Grandma, we put pictures of our children on a photo album web site such as Flickr. Then not only can Grandma see them—so can Grandma's friends and anyone else. So what? They are cute and harmless. But suppose a tourist takes a vacation snapshot and you just happen to appear in the background, at a restaurant where no one knew you were dining. If the tourist uploads his photo, the whole world could know where you were, and when you were there.

Data leaks. Credit card records are supposed to stay locked up in a data warehouse, but escape into the hands of identity thieves. And we sometimes give information away just because we get something back for doing so. A company will give you free phone calls to anywhere in the world—if you don't mind watching ads for the products its computers hear you talking about.

And those are merely things that are happening today. The explosion, and the social disruption it will create, have barely begun.

We already live in a world in which there is enough memory *just in digital cameras* to store every word of every book in the Library of Congress a hundred times over. So much email is being sent that it could transmit the full text of the Library of Congress in ten minutes. Digitized pictures and sounds take more space than words, so emailing all the images, movies, and sounds might take a year—but that is just today. The explosive growth is still happening. Every year we can store more information, move it more quickly, and do far more ingenious things with it than we could the year before.

So much disk storage is being produced every year that it could be used to record a page of information, every minute or two, about you *and every other human being on earth*. A remark made long ago can come back to haunt a political candidate, and a letter jotted quickly can be a key discovery for a

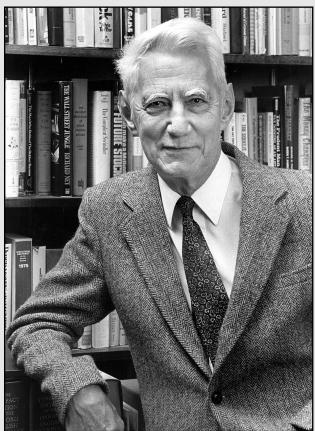
biographer. Imagine what it would mean to record every word every human being speaks or writes in a lifetime. The technological barrier to that has already been removed: There is enough storage to remember it all. Should any social barrier stand in the way?

Sometimes things seem to work both better and worse than they used to. A “public record” is now *very* public—before you get hired in Nashville, Tennessee, your employer can figure out if you were caught ten years ago taking an illegal left turn in Lubbock, Texas. The old notion of a “sealed court record” is mostly a fantasy in a world where any tidbit of information is duplicated, cataloged, and moved around endlessly. With hundreds of TV and radio stations and millions of web sites, Americans love the variety of news sources, but are still adjusting uncomfortably to the displacement of more authoritative sources. In China, the situation is reversed: The technology creates greater government control of the information its citizens receive, and better tools for monitoring their behavior.

This book is about how the digital explosion is changing everything. It explains the technology itself—why it creates so many surprises and why things often don’t work the way we expect them to. It is also about things the information explosion is destroying: old assumptions about our privacy, about our identity, and about who is in control of our lives. It’s about how we got this way, what we are losing, and what remains that society still has a chance to put right. The digital explosion is creating both opportunities and risks. Many of both will be gone in a decade, settled one way or another. Governments, corporations, and other authorities are taking advantage of the chaos, and most of us don’t even see it happening. Yet we all have a stake in the outcome. Beyond the science, the history, the law, and the politics, this book is a wake-up call. The forces shaping your future are digital, and you need to understand them.

The Koans of Bits

Bits behave strangely. They travel almost instantaneously, and they take almost no space to store. We have to use physical metaphors to make them understandable. We liken them to dynamite exploding or water flowing. We even use social metaphors for bits. We talk about two computers agreeing on some bits, and about people using burglary tools to steal bits. Getting the right metaphor is important, but so is knowing the limitations of our metaphors. An imperfect metaphor can mislead as much as an apt metaphor can illuminate.



CLAUDE SHANNON

Claude Shannon (1916–2001) is the undisputed founding figure of information and communication theory. While working at Bell Telephone Laboratories after the Second World War, he wrote the seminal paper, "A mathematical theory of communication," which foreshadowed much of the subsequent development of digital technologies.

Published in 1948, this paper gave birth to the now-universal realization that the bit is the natural unit of information, and to the use of the term.

Alcatel-Lucent, http://www.bell-labs.com/news/2001/february/26/shannon2_lg.jpeg.

We offer seven truths about bits. We call them “koans” because they are paradoxes, like the Zen verbal puzzles that provoke meditation and enlightenment. These koans are oversimplifications and over-generalizations. They describe a world that is developing but hasn’t yet fully emerged. But even today they are truer than we often realize. These themes will echo through our tales of the digital explosion.

Koan 1: It’s All Just Bits

Your computer successfully creates the illusion that it contains photographs, letters, songs, and movies. All it really contains is bits, lots of them, patterned in ways you can’t see. Your computer was designed to store just bits—all the files and folders and different kinds of data are illusions created by computer programmers. When you send an email containing a photograph, the computers that handle your message as it flows through the Internet have no idea that what they are handling is part text and part graphic. Telephone calls are also just bits, and that has helped create competition—traditional phone companies, cell phone companies, cable TV companies, and Voice over IP (VoIP) service providers can just shuffle bits around to each other to complete calls. The Internet was designed to handle just bits, not emails or attachments, which are inventions of software engineers. We couldn’t live without those more intuitive concepts, but they are artifices. Underneath, it’s all just bits.

This koan is more consequential than you might think. Consider the story of NARAL Pro-Choice America and Verizon Wireless. NARAL wanted to form a

text messaging group to send alerts to its members. Verizon decided not to allow it, citing the “controversial or unsavory” things the messages might contain. Text message alert groups for political candidates it would allow, but not for political causes it deemed controversial. Had NARAL simply wanted telephone service or an 800 number, Verizon would have had no choice. Telephone companies were long ago declared “common carriers.” Like railroads, phone companies are legally prohibited from picking and choosing customers from among those who want their services. In the bits world, there is no difference between a text message and a wireless phone call. It’s all just bits, traveling through the air by radio waves. But the law hasn’t caught up to the technology. It doesn’t treat all bits the same, and the common carriage rules for voice bits don’t apply to text message bits.

EXCLUSIVE AND RIVALROUS

Economists would say that bits, unless controlled somehow, tend to be *non-exclusive* (once a few people have them, it is hard to keep them from others) and *non-rivalrous* (when someone gets them from me, I don’t have any less). In a letter he wrote about the nature of ideas, Thomas Jefferson eloquently stated both properties. *If nature has made any one thing less susceptible than all others of exclusive property, it is the action of the thinking power called an idea, which an individual may exclusively possess as long as he keeps it to himself; but the moment it is divulged, it forces itself into the possession of every one, and the receiver cannot dispossess himself of it. Its peculiar character, too, is that no one possesses the less, because every other possesses the whole of it.*

Verizon backed down in the case of NARAL, but not on the principle. A phone company can do whatever it thinks will maximize its profits in deciding whose messages to distribute. Yet no sensible engineering distinction can be drawn between text messages, phone calls, and any other bits traveling through the digital airwaves.

Koan 2: Perfection Is Normal

To err is human. When books were laboriously transcribed by hand, in ancient scriptoria and medieval monasteries, errors crept in with every copy. Computers and networks work differently. Every copy is perfect. If you email a photograph to a friend, the friend won’t receive a fuzzier version than the original. The copy will be identical, down to the level of details too small for the eye to see.

Computers do fail, of course. Networks break down too. If the

power goes out, nothing works at all. So the statement that copies are normally perfect is only relatively true. Digital copies are perfect only to the extent that they can be communicated at all. And yes, it is possible in theory that a single bit of a big message will arrive incorrectly. But networks don't just pass bits from one place to another. They check to see if the bits seem to have been damaged in transit, and correct them or retransmit them if they seem incorrect. As a result of these error detection and correction mechanisms, the odds of an actual error—a character being wrong in an email, for example—are so low that we would be wiser to worry instead about a meteor hitting our computer, improbable though precision meteor strikes may be.

The phenomenon of perfect copies has drastically changed the law, a story told in Chapter 6, “Balance Toppled.” In the days when music was distributed on audio tape, teenagers were not prosecuted for making copies of songs, because the copies weren’t as good as the originals, and copies of copies would be even worse. The reason that thousands of people are today receiving threats from the music and movie industries is that their copies are perfect—not just as good as the original, but identical to the original, so that even the notion of “original” is meaningless. The dislocations caused by file sharing are not over yet. The buzzword of the day is “intellectual property.” But bits are an odd kind of property. Once I release them, everybody has them. And if I give you my bits, I don’t have any fewer.

Koan 3: There Is Want in the Midst of Plenty

Vast as world-wide data storage is today, five years from now it will be ten times as large. Yet the information explosion means, paradoxically, the loss of information that is not online. One of us recently saw a new doctor at a clinic he had been using for decades. She showed him dense charts of his blood chemistry, data transferred from his home medical device to the clinic’s computer—more data than any specialist could have had at her disposal five years ago. The doctor then asked whether he had ever had a stress test and what the test had shown. Those records should be all there, the patient explained, in the medical file. But it was in the *paper* file, to which the doctor did not have access. It wasn’t in the *computer’s* memory, and the patient’s memory was being used as a poor substitute. The old data might as well not have existed at all, since it wasn’t digital.

Even information that exists in digital form is useless if there are no devices to read it. The rapid progress of storage engineering has meant that data stored on obsolete devices effectively ceases to exist. In Chapter 3, “Ghosts in the Machine,” we shall see how a twentieth-century update of the

eleventh-century British Domesday Book was useless by the time it was only a sixtieth the age of the original.

Or consider search, the subject of Chapter 4, “Needles in the Haystack.” At first, search engines such as Google and Yahoo! were interesting conveniences, which a few people used for special purposes. The growth of the World Wide Web has put so much information online that search engines are for many people the first place to look for something, before they look in books or ask friends. In the process, appearing prominently in search results has become a matter of life or death for businesses. We may move on to purchase from a competitor if we can’t find the site we wanted in the first page or two of results. We may assume something didn’t happen if we can’t find it quickly in an online news source. If it can’t be found—and found quickly—it’s just as though it doesn’t exist at all.

Koan 4: Processing Is Power

MOORE'S LAW

Gordon Moore, founder of Intel Corporation, observed that the density of integrated circuits seemed to double every couple of years. This observation is referred to as “Moore’s Law.” Of course, it is not a natural law, like the law of gravity. Instead, it is an empirical observation of the progress of engineering and a challenge to engineers to continue their innovation. In 1965, Moore predicted that this exponential growth would continue for quite some time. That it has continued for more than 40 years is one of the great marvels of engineering. No other effort in history has sustained anything like this growth rate.

The speed of a computer is usually measured by the number of basic operations, such as additions, that can be performed in one second. The fastest computers available in the early 1940s could perform about five operations per second. The fastest today can perform about a trillion. Buyers of personal computers know that a machine that seems fast today will seem slow in a year or two.

For at least three decades, the increase in processor speeds was exponential. Computers became twice as fast every couple of years. These increases were one consequence of “Moore’s Law” (see sidebar).

Since 2001, processor speed has not followed Moore’s Law; in fact, processors have hardly grown faster

at all. But that doesn’t mean that computers won’t continue to get faster. New chip designs include multiple processors on the same chip so the work can be split up and performed in parallel. Such design innovations promise to

achieve the same effect as continued increases in raw processor speed. And the same technology improvements that make computers faster also make them cheaper.

The rapid increase in processing power means that inventions move out of labs and into consumer goods very quickly. Robot vacuum cleaners and self-parking vehicles were possible in theory a decade ago, but now they have become economically feasible. Tasks that today seem to require uniquely human skills are the subject of research projects in corporate or academic laboratories. Face recognition and voice recognition are poised to bring us new inventions, such as telephones that know who is calling and surveillance cameras that don't need humans to watch them. The power comes not just from the bits, but from being able to do things with the bits.

Koan 5: More of the Same Can Be a Whole New Thing

Explosive growth is exponential growth—doubling at a steady rate. Imagine earning 100% annual interest on your savings account—in 10 years, your money would have increased more than a thousandfold, and in 20 years, more than a millionfold. A more reasonable interest rate of 5% will hit the same growth points, just 14 times more slowly. Epidemics initially spread exponentially, as each infected individual infects several others.

When something grows exponentially, for a long time it may seem not to be changing at all. If we don't watch it steadily, it will seem as though something discontinuous and radical occurred while we weren't looking.

That is why epidemics at first go unnoticed, no matter how catastrophic they may be when full-blown. Imagine one sick person infecting two healthy people, and the next day each of those two infects two others, and the next day after that each of those four infects two others, and so on. The number of newly infected each day grows from two to four to eight. In a week, 128 people come down with the disease in a single day, and twice that number are now sick, but in a population of ten million, no one notices. Even after two weeks, barely three people in a thousand are sick. But after another week, 40% of the population is sick, and society collapses.

Exponential growth is actually smooth and steady; it just takes very little time to pass from unnoticeable change to highly visible. Exponential growth of anything can suddenly make the world look utterly different than it had been. When that threshold is passed, changes that are “just” quantitative can look qualitative.

Another way of looking at the apparent abruptness of exponential growth—its explosive force—is to think about how little lead time we have to respond to it. Our hypothetical epidemic took three weeks to overwhelm the

population. At what point was it only a half as devastating? The answer is *not* “a week and a half.” The answer is *on the next to last day*. Suppose it took a week to develop and administer a vaccine. Then noticing the epidemic after a week and a half would have left ample time to prevent the disaster. But that would have required understanding that there *was* an epidemic when only 2,000 people out of ten million were infected.

The information story is full of examples of unperceived changes followed by dislocating explosions. Those with the foresight to notice the explosion just a little earlier than everyone else can reap huge benefits. Those who move a little too slowly may be overwhelmed by the time they try to respond. Take the case of digital photography.

In 1983, Christmas shoppers could buy digital cameras to hook up to their IBM PC and Apple II home computers. The potential was there for anyone to see; it was not hidden in secret corporate laboratories. But digital photography did not take off. Economically and practically, it couldn’t. Cameras were too bulky to put in your pocket, and digital memories were too small to hold many images. Even 14 years later, film photography was still a robust industry. In early 1997, Kodak stock hit a record price, with a 22% increase in quarterly profit, “fueled by healthy film and paper sales...[and] its motion picture film business,” according to a news report. The company raised its dividend for the first time in eight years. But by 2007, digital memories had become huge, digital processors had become fast and compact, and both were cheap. As a result, cameras had become little computers. The company that was once synonymous with photography was a shadow of its former self. Kodak announced that its employee force would be cut to 30,000, barely a fifth the size it was during the good times of the late 1980s. The move would cost the company more than \$3 billion. Moore’s Law moved faster than Kodak did.

In the rapidly changing world of bits, it pays to notice even small changes, and to do something about them.

Koan 6: Nothing Goes Away

2,000,000,000,000,000,000,000.

That is the number of bits that were created and stored away in 2007, according to one industry estimate. The capacity of disks has followed its own version of Moore’s Law, doubling every two or three years. For the time being at least, that makes it possible to save everything though recent projections suggest that by 2011, we may be producing more bits than we can store.

In financial industries, federal laws now *require* massive data retention, to assist in audits and investigations of corruption. In many other businesses, economic competitiveness drives companies to save everything they collect and to seek out new data to retain. Wal-Mart stores have tens of millions of transactions every day, and every one of them is saved—date, time, item, store, price, who made the purchase, and how—credit, debit, cash, or gift card. Such data is so valuable to planning the supply chain that stores will pay money to get more of it from their customers. That is really what supermarket loyalty cards provide—shoppers are supposed to think that the store is granting them a discount in appreciation for their steady business, but actually the store is paying them for information about their buying patterns. We might better think of a privacy tax—we pay the regular price *unless* we want to keep information about our food, alcohol, and pharmaceutical purchases from the market; to keep our habits to ourselves, we pay extra.

The massive databases challenge our expectations about what will happen to the data about us. Take something as simple as a stay in a hotel. When you check in, you are given a keycard, not a mechanical key. Because the keycards can be deactivated instantly, there is no longer any great risk associated with losing your key, as long as you report it missing quickly. On the other hand, the hotel now has a record, accurate to the second, of every time you entered your room, used the gym or the business center, or went in the back door after-hours. The same database could identify every cocktail and steak you charged to the room, which other rooms you phoned and when, and the brands of tampons and laxatives you charged at the hotel's gift shop. This data might be merged with billions like it, analyzed, and transferred to the parent company, which owns restaurants and fitness centers as well as hotels. It might also be lost, or stolen, or subpoenaed in a court case.

The ease of storing information has meant asking for more of it. Birth certificates used to include just the information about the child's and parents' names, birthplaces, and birthdates, plus the parents' occupations. Now the electronic birth record includes how much the mother drank and smoked during her pregnancy, whether she had genital herpes or a variety of other medical conditions, and both parents' social security numbers. Opportunities for research are plentiful, and so are opportunities for mischief and catastrophic accidental data loss.

And the data will all be kept forever, unless there are policies to get rid of it. For the time being at least, the data sticks around. And because databases are intentionally duplicated—backed up for security,

The data will all be kept forever, unless there are policies to get rid of it.

or shared while pursuing useful analyses—it is far from certain that data can ever be permanently expunged, even if we wish that to happen. The Internet consists of millions of interconnected computers; once data gets out, there is no getting it back. Victims of identity theft experience daily the distress of having to remove misinformation from the record. It seems never to go away.

Koan 7: Bits Move Faster Than Thought

The Internet existed before there were personal computers. It predates the fiber optic communication cables that now hold it together. When it started around 1970, the ARPANET, as it was called, was designed to connect a handful of university and military computers. No one imagined a network connecting tens of millions of computers and shipping information around the world in the blink of an eye. Along with processing power and storage capacity, networking has experienced its own exponential growth, in number of computers interconnected and the rate at which data can be shipped over long distances, from space to earth and from service providers into private homes.

The Internet has caused drastic shifts in business practice. Customer service calls are outsourced to India today not just because labor costs are low there. Labor costs have *always* been low in India, but international telephone calls used to be expensive. Calls about airline reservations and lingerie returns are answered in India today because it now takes almost no time and costs almost no money to send to India the bits representing your voice. The same principle holds for professional services. When you are X-rayed at your local hospital in Iowa, the radiologist who reads the X-ray may be half a world away. The digital X-ray moves back and forth across the world faster than a physical X-ray could be moved between floors of the hospital. When you place an order at a drive-through station at a fast food restaurant, the person taking the order may be in another state. She keys the order so it appears on a computer screen in the kitchen, a few feet from your car, and you are none the wiser. Such developments are causing massive changes to the global economy, as industries figure out how to keep their workers in one place and ship their business as bits.

In the bits world, in which messages flow instantaneously, it sometimes seems that distance doesn't matter at all. The consequences can be startling. One of us, while dean of an American college, witnessed the shock of a father receiving condolences on his daughter's death. The story was sad but familiar, except that this version had a startling twist. Father and daughter were

both in Massachusetts, but the condolences arrived from half-way around the world before the father had learned that his daughter had died. News, even the most intimate news, travels fast in the bits world, once it gets out. In the fall of 2007, when the government of Myanmar suppressed protests by Buddhist monks, television stations around the world showed video clips taken by cell phone, probably changing the posture of the U.S. government. The Myanmar rebellion also shows the power of information control when information is just bits. The story dropped off the front page of the newspapers once the government took total control of the Internet and cell phone towers.

The instantaneous communication of massive amounts of information has created the misimpression that there is a place called “Cyberspace,” a land without frontiers where all the world’s people can be interconnected as though they were residents of the same small town. That concept has been decisively refuted by the actions of the world’s courts. National and state borders still count, and count a lot. If a book is bought online in England, the publisher and author are subject to British libel laws rather than those of the homeland of the author or publisher. Under British law, defendants have to prove their innocence; in the U.S., plaintiffs have to prove the guilt of the defendants. An ugly downside to the explosion of digital information and its movement around the world is that information may become less available even where it would be legally protected (we return to this subject in Chapter 7, “You Can’t Say That on the Internet”). Publishers fear “libel tourism”—lawsuits in countries with weak protection of free speech, designed to intimidate authors in more open societies. It may prove simpler to publish only a single version of a work for sale everywhere, an edition omitting information that might somewhere excite a lawsuit.

Good and Ill, Promise and Peril

The digital explosion has thrown a lot of things up for grabs and we all have a stake in who does the grabbing. The way the technology is offered to us, the way we use it, and the consequences of the vast dissemination of digital information are matters not in the hands of technology experts alone. Governments and corporations and universities and other social institutions have a say. And ordinary citizens, to whom these institutions are accountable, can influence their decisions. Important choices are made every year, in government offices

and legislatures, in town meetings and police stations, in the corporate offices of banks and insurance companies, in the purchasing departments of chain stores and pharmacies. We all can help raise the level of discourse and understanding. We can all help ensure that technical decisions are taken in a context of ethical standards.

We offer two basic morals. The first is that information technology is inherently neither good nor bad—it can be used for good or ill, to free us or to shackle us. Second, new technology brings social change, and change comes with both risks and opportunities. All of us, and all of our public agencies and private institutions, have a say in whether technology will be used for good or ill and whether we will fall prey to its risks or prosper from the opportunities it creates.

Technology Is Neither Good nor Bad

Any technology can be used for good or ill. Nuclear reactions create electric power and weapons of mass destruction. The same encryption technology that makes it possible for you to email your friends with confidence that no eavesdropper will be able to decipher your message also makes it possible for terrorists to plan their attacks undiscovered. The same Internet technology that facilitates the widespread distribution of educational works to impoverished students in remote locations also enables massive copyright infringement. The photomanipulation tools that enhance your snapshots are used by child pornographers to escape prosecution.

The key to managing the ethical and moral consequences of technology while nourishing economic growth is to *regulate the use* of technology without *banning or restricting its creation*.

It is a marvel that anyone with a smart cell phone can use a search engine to get answers to obscure questions almost anywhere. Society is rapidly being freed from the old limitations of geography and status in accessing information.

The same technologies can be used to monitor individuals, to track their behaviors, and to control what information they receive. Search engines need not return unbiased results. Many users of web browsers do not realize that the sites they visit may archive their actions. Technologically, there could be a record of exactly what you have been accessing and when, as you browse a library or bookstore catalog, a site selling pharmaceuticals, or a service offering advice on contraception or drug overdose. There are vast opportunities to

use this information for invasive but relatively benign purposes, such as marketing, and also for more questionable purposes, such as blacklisting and blackmail. Few regulations mandate disclosure that the information is being collected, or restrict the use to which the data can be put. Recent federal laws, such as the USA PATRIOT Act, give government agencies sweeping authority to sift through mostly innocent data looking for signs of “suspicious activity” by potential terrorists—and to notice lesser transgressions, such as Governor Spitzer’s, in the process. Although the World Wide Web now reaches into millions of households, the rules and regulations governing it are not much better than those of a lawless frontier town of the old West.

BLACKLISTS AND WHITELISTS

In the bits world, providers of services can create blacklists or whitelists. No one on a blacklist can use the service, but everyone else can. For example, an auctioneer might put people on a blacklist if they did not pay for their purchases. But service providers who have access to other information about visitors to their web sites might use undisclosed and far more sweeping criteria for blacklisting. A whitelist is a list of parties to whom services are available, with everyone else excluded. For example, a newspaper may whitelist its home delivery subscribers for access to its online content, allowing others onto the whitelist only after they have paid.

New Technologies Bring Both Risks and Opportunities

The same large disk drives that enable anyone with a home computer to analyze millions of baseball statistics also allow anyone with access to confidential information to jeopardize its security. Access to aerial maps via the Internet makes it possible for criminals to plan burglaries of upscale houses, but technologically sophisticated police know that records of such queries can also be used to solve crimes.

Even the most un-electronic livelihoods are changing because of instant worldwide information flows. There are no more pool hustlers today—journeymen wizards of the cue, who could turn up in pool halls posing as out-of-town bumpkins just looking to bet on a friendly game, and walk away with big winnings. Now when any newcomer comes to town and cleans up, his name and face are on AZBilliards.com instantly for pool players everywhere to see.

Social networking sites such as facebook.com, myspace.com, and match.com have made their founders quite wealthy. They have also given birth to many thousands of new friendships, marriages, and other ventures. But those pretending to be your online friends may not be as they seem. Social networking has made it easier for predators to take advantage of the naïve, the lonely, the elderly, and the young.

In 2006, a 13-year-old girl, Megan Meier of Dardenne Prairie, Missouri, made friends online with a 16-year-old boy named “Josh.” When “Josh” turned against her, writing “You are a bad person and everybody hates you.... The world would be a better place without you,” Megan committed suicide. Yet Josh did not exist. Josh was a MySpace creation—but of whom? An early police report stated that the mother of another girl in the neighborhood acknowledged “instigating” and monitoring the account. That woman’s lawyer later blamed someone who worked for his client. Whoever may have sent the final message to Megan, prosecutors are having a hard time identifying any law that might have been broken. “I can start MySpace on every single one of you and spread rumors about every single one of you,” said Megan’s mother, “and what’s going to happen to me? Nothing.”

Along with its dazzling riches and vast horizons, the Internet has created new manifestations of human evil—some of which, including the cyber-harassment Megan Meier suffered, may not be criminal under existing law. In a nation deeply committed to free expression as a legal right, which Internet evils should be crimes, and which are just wrong?

Vast data networks have made it possible to move work to where the people are, not people to the work. The results are enormous business opportunities for entrepreneurs who take advantage of these technologies and new enterprises around the globe, and also the other side of the coin: jobs lost to outsourcing.

The difference every one of us can make, to our workplace or to another institution, can be to ask a question at the right time about the risks of some new technological innovation—or to point out the possibility of doing something in the near future that a few years ago would have been utterly impossible.



We begin our tour of the digital landscape with a look at our privacy, a social structure the explosion has left in shambles. While we enjoy the benefits of ubiquitous information, we also sense the loss of the shelter that privacy once gave us. And we don't know what we want to build in its place. The good and ill of technology, and its promise and peril, are all thrown together when information about us is spread everywhere. In the post-privacy world, we stand exposed to the glare of noonday sunlight—and sometimes it feels strangely pleasant.

Introduction to Abstraction

This activity introduces the concept that abstractions built upon binary sequences can be used to represent all digital data. It also introduces the idea the computing has global impacts. It focuses, in part, on the following learning objectives:

- The student can develop an abstraction.

Introduction

In this course, there's both a practical component, such as using SNAP/BYOB in the lab, and a "big ideas" component, in lecture and discussion. One of the things we want you to take away from this course is to know that there's more to computer science than just writing computer programs, although of course we use those other things to help in writing programs. So we'll talk about some aspects of the social context of computing, some of the theoretical explorations of the limits of computation, and some important moments in the history of computer science – for example, you'll learn in a few weeks how one of the first computer scientists more or less single-handedly won World War II for the good guys.

Abstraction

Abstraction is arguably the central idea of all of computer science. Computer programming is easy, as long as the programs are small. What's hard isn't the programming, but the keeping track of details in a huge program. The solution is **chunking**, or **layering** – two metaphors for abstraction.

Practical Abstraction Example

The classic example is thinking about a car. Cars are made of nuts, bolts, metal rods, big metal blocks, rubber or paper gaskets, plastic containers for fluids, rivets, wires, and so on. (Each piece of metal is further made of atoms, which are made of electrons, protons, and neutrons, which are made of quarks, and so on down.) But if you're trying to repair a car, you don't think in those terms; if you did, you'd never find where the problem is. Instead you think about the engine, the alternator, the fuel injectors, the brakes, the transmission, and so on. *That's* abstraction.

The march of technological progress is, at least in part, a march toward greater and greater abstraction. *Each step reduces the extent to which people have to think about details.* Sticking with cars as the example, in the early days, every driver had to be at least something of a mechanic, knowing how to deal with the rather frequent failures of the machinery. Before automatic transmissions, only people with some understanding of gear ratios could drive. (In the really early days, they couldn't downshift without mastering the skill of double-clutching.)

The automatic transmission made possible an enormous abstraction. All the complexity of the machinery that makes a car work was hidden under the surface of a very simple model: You push the pedal on the right and the car speeds up; you push the pedal on the left and it slows down. Suddenly just about anyone could drive a car.

Of course, the widespread use of cars has turned out to be a mixed blessing. Cars are one of the main causes of pollution and global warming. Computers, too, have their downsides, which we'll be discussing later. Many historians of science stay away from the word "progress," which I used two paragraphs back, because of its implicit suggestion that the development of new technology is always good. But before we can criticize technology we should understand something about how it works, and abstraction is a very powerful organizing idea to describe the mechanism.

Those two pedals, the gas pedal and the brake, are an **interface**, also known as an **abstraction barrier**. On the driver's side of the abstraction, what matters is the *behavior* provided by this interface. Push this one to speed up, that one to slow down. Once that interface became standardized, further technical development has dramatically changed what happens up in the engine compartment. Originally, the gas pedal mechanically pushed a lever controlling a valve that determined the rate at which gasoline could flow into the engine. More gasoline, bigger explosions inside the engine, more power, so more speed. Today, the gas pedal doesn't really do anything mechanically, except provide an input to a computer inside the car, whose job is to control the fuel injection system. Your input is combined with other information about the car's environment to operate smaller valves, one per cylinder, that control the gas/air mixture more precisely.

The brake pedal has had a similar history. Originally, your foot directly provided the power to push the brake pads against the wheels. Then a new mechanism was developed, preserving the interface – push here to slow down – but now using the pressure from your foot to operate a hydraulic system that does the hard work of pressing the pads against the wheels. But there was one important difference. The first “power brakes,” like the modern gas pedal, completely eliminated the mechanical linkage between the pedal and the actual brakes. But after a few accidents in which people couldn't stop their cars because the engine died, this design was modified. Today you have “power assisted brakes,” which means that your foot both operates a hydraulic cylinder and also directly puts pressure on the brakes. If the engine fails, you have to push a lot harder to stop the car, but at least it's possible. The latest development, anti-lock brakes, actually lets a computer in the car override your pressure on the brake pedal if you are in danger of putting the car into a skid by trying to stop too abruptly.

But the point is that drivers who aren't particularly interested in cars don't have to know any of this. All they have to know is that you push the pedal on the right to speed up, and the one on the left to slow down! This *interface* has survived through several generations of underlying technology, because it's a good interface – simple but expressive. Car engineers could have made each generation of new technology more visible to drivers, with lots of knobs and switches and readouts, but they wisely refrained, and stayed with the abstraction – the interface – developed a century ago.

Abstraction in Computer Programming

How does abstraction work in computer programming? We'll be revisiting this question all semester. But, for a starting point, think about the blocks in the BYOB menu. Each block names a simple intention, comparable to “speed up” or “slow down” in a car. But the mechanisms that make those blocks do their job are actually doing very complicated detail work. For example, “move 10 steps,” above the abstraction barrier, just tells Scratch to move over a little. But BYOB isn't a real image; it's a collection of colored dots drawn on a computer screen. “Move 10 steps” really means to erase each of those dots, then redraw them all in a different position. And the “glide” block is even more complicated; it has to erase and redraw Scratch many times, moving just a tiny bit each time.

But if the abstraction is working well, you're not thinking about tiny dots of light at all, while working with Scratch/BYOB. You're moving a cat!

We will begin with BYOB, a visual programming language. You can take a complicated sequence of actions, wrap them up, and present them to another user – or to yourself thinking at a higher level of abstraction – as a new interface, allowing you to command a new behavior without having to know anything about the detailed implementation.

Material from Dr. Brian Harvey, UC Berkeley

More on Abstraction

This introduces the concept that *abstractions built upon binary sequences can be used to represent all digital data*. It also introduces the idea the *computing has global impacts*. It focuses, in part, on the following learning objectives:

- 5: The student can describe the combination of abstractions used to represent data.
- 6: The student can explain how binary sequences are used to represent digital data.
- 28: The student can analyze how computing affects communication, interaction, and cognition.
- 29: The student can connect computing with innovations in other fields.
- 30: The student can analyze the beneficial and harmful effects of computing.
- 31: The student can connect computing within economic, social, and cultural contexts.

Bits as the Natural Unit of Information

This was the seminal insight of Claude Shannon (http://en.wikipedia.org/wiki/Claude_Shannon), founder of information and communication theory. As this chapter illustrates all information stored on computers (documents, photos, songs, etc.) and communicated through the Internet (email, blogs, twitter posts, etc.) are represented as bits, zeros and ones.

Digital data are represented in discrete *bits* (Binary digITs). Analog data are represented as a continuous quantity. Think of the difference between a digital watch that displays hours, minutes, seconds, and tenths of a second, and an *analog clock*, with hour hand, minute hand, and a sweep second hand.

Think of the difference in *fidelity* between digital vs. analog recordings (<http://www.howstuffworks.com/analog-digital3.htm>).

Moore's Law

Moore's law (http://en.wikipedia.org/wiki/Moore%27s_law) is the observation that the number of transistors that could be packed onto a integrated circuit seemed to double every two years or so. This remarkable trend about exponential growth (<http://en.wikipedia.org/wiki/File:Exponential.svg>) in chip density has proved to be true for 40 years now.

Someone offers you a summer job and offers you two payment schemes: (1) \$10 per hour for 40 hours per week for 30 days or (2) One cent on day 1, two cents and day two, four cents on day three and on (doubling each day) for 30 days. Which pay would you choose? Click here to see how exponential growth affects this question (<http://www.cs.trincoll.edu/~ram/aitalk/twos.txt>).

It's All Just Bits

Binary digits (bits) correspond naturally to electronic circuits where 1 represents 'on' and 0 represents 'off'. In computers, binary sequences are used to represent all kinds of data: text, numbers, images, sounds, ..., everything.

Depending on the context, the same string of bits can represent different types of information. These are all examples of **abstraction** at work:

- The sequence `0100 0001` can represent the binary numeral (http://en.wikipedia.org/wiki/Binary_numeral_system) for the decimal value 65 when it occurs calculator.
- The sequence `0100 0001` can represent the ASCII letter (<http://www.ascii.cl/>) 'A' when it occurs in an email message.

- The sequence `0100 0001` can represent the amount of Red in the RGB color model (http://en.wikipedia.org/wiki/RGB_color_model) when it occurs in an image.
- The sequence `0100 0001` can represent the MOV B,C machine operation on the Intel 8080 8-bit processor (<http://nemesis.lonestar.org/computers/tandy/software/apps/m4/qd/opcodes.html>)

A Little Bit About Bits

A **byte** is an 8-bit sequence. Historically an 8-bit byte was used to represent a single character in computer memory.

The length of a binary sequence – a sequence of 0s and 1s – determines the number of different sequences that can be generated and therefore the number of different things that can be represented by such a sequence.

For example, with 1 bit, you can have two different sequences, 0 or 1, which can stand for two different colors, say, 0 stands for white and 1 for black. With 2 bits, you can have four different sequences, 00, 01, 10, or 11, which can represent four different colors, white, black, red, green. And so on.

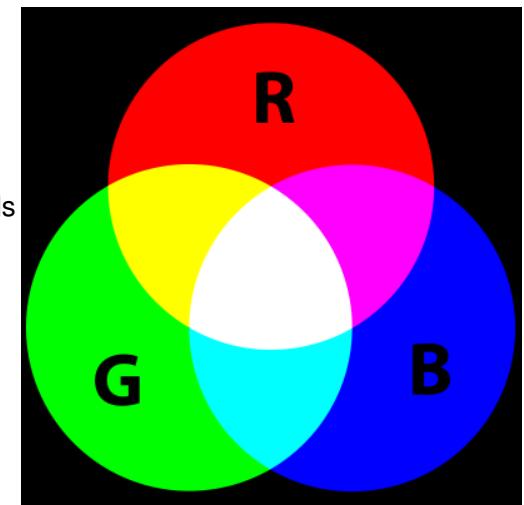
In general, an n-bit sequence can represent 2^n different things. Here's how:

Number of Bits	Number of Things	Representations
1	2 (ie 2^1)	0,1
2	4 (ie 2^2)	00,01,10,11
3	8 (ie 2^3)	000,001,010,011,100,101,110,111
4	16 (ie 2^4)	
5	32 (ie 2^5)	
6	64 (ie 2^6)	
7	128 (ie 2^7)	
8	256 (ie 2^8)	

Data Abstraction: How Colors Are Represented in Bits

The RGB model (http://en.wikipedia.org/wiki/RGB_color_model) adds together 3 primary colors, Red, Green, and Blue, where each color component (R,G,B) is represented as an 8-bit **byte**.

If there are 256 (2^8) possible values for each of R,G,B, the triplet can represent $256 \times 256 \times 256 = 16,777,216$ different colors, which corresponds well to the number of colors the human eye can distinguish.



- (R, G, B)
- (65,65,65) = Grey
- (255,0,0) = Red
- (0,255,0) = Green
- (0,0,255) = Blue
- (65,0,0) = Brown

Try to mix your own colors: [Colorschemer.com](http://www.colorschemer.com/online.html) (<http://www.colorschemer.com/online.html>)

Perfection is Normal

Because they are based on strings of discrete bits, *digital* (as opposed to *analog*) copies are perfect copies.

Computer scientists and engineers have developed effective error detection (http://en.wikipedia.org/wiki/Parity_bit) and error correction schemes to insure accurate data representation and communication.

Example: Parity Bit Error Detection

Suppose you are sending a stream of data to a server. By adding a parity bit, you enable the server to detect some basic transmission errors. For example, if the server expects that every byte will contain an **even number of 1s** and it detects a byte such as `0001 0101` with an odd number of 1s, it can tell that an error occurred. Perhaps the user meant to send `0000 0101` but one of the bits was flipped from 0 to 1 during transmission.

A **parity bit** is a bit that is added as the leftmost bit of a bit string to ensure that the number of bits that are 1 in the bit string are even or odd.

To see how this works, suppose our data are stored in strings containing 7 bits.

In an **even parity scheme** the eighth bit, the parity bit, is set to 1 if the number of 1s in the 7 data bits is odd, thereby making the number of 1s in the 8-bit byte an even number. It is set to 0 if the number of 1s in the data is even.

In an **odd parity scheme** the eighth bit, the parity bit, is set to 1 if the number of 1s in the 7 data bits is even, thereby making the number of 1s in the 8-bit byte an odd number. It is set to 0 if the number of 1s in the data is odd.

The following table summarizes this approach.

Data Bits (7)	Even Parity (even number 1s)	Odd Parity (odd number 1s)
000 0000 (0 1s)	0000 0000	1000 0000
011 0010 (3 1s)	1011 0010	0011 0010
011 0011 (4 1s)	0011 0011	1011 0011
011 0111 (5 1s)	1011 0111	0011 0111

Question: What would happen in this scheme if 2 bits were switched from 1 to 0 or 0 to 1?

Quiz Yourself

To see if you understand these concepts, try the following quizzes. If you can get ten-in-a-row correct, that's a pretty good indication that you get it.

- Parity Error Detection I (<http://www.cs.trincoll.edu/%7Eram/q/110/parity-error-detection.html>)
- Parity Error Detection II (<http://www.cs.trincoll.edu/%7Eram/q/110/parity-error-detection-2.html>)

Variables and Abstraction

When you program you will create variables to make the script much more functional – i.e., the user could change the value of the variable.

This is an important example of **abstraction** at work – in this case, we are letting an *abstract symbol* (the variable) represent or stand for something else (its value, 2 or 8).

What is a Variable?

A *variable* in a computer program is a *symbol* that represents a *memory location* where a piece of data can be stored. You can think of a computer memory as a large array of numbered mail boxes, where the numbers represent the address of the memory location. In this example, there are 8 memory locations numbered 17-24 (in decimal) and 10001 - 11000 (in binary). None of the locations have any value stored in them yet.

17	18	19	20	21	22	23	24
10001	10010	10011	10100	10101	10110	10111	11000

When you define a *variable*, you are giving name to a memory location.

name	score	wins	20	21	22	23	24
10001	10010	10011	10100	10101	10110	10111	11000
Joe	8	2					

What is a Value?

In a computer program a *value* is a symbol of a piece of *data*. For example, the numeral ‘8’ represents the number 8. Values (data) are stored in the computer’s memory locations.

Symbols like the numeral ‘8’ are also examples of *abstractions*. The numeral ‘8’ and the word ‘eight’ and the binary string ‘1000’ are all symbols that represent the number 8, which is itself an abstract concept of 8 things. For example, they can all be used to refer to the number of little circles here: o o o o o o o o.

So, in a computer program we use abstract symbols to represent both *variables* and *values*.

Values vs. Variables

It’s important to distinguish between the variable’s *name* (e.g., score or name) from the *value* that it represents – i.e., from the value that it is storing in its memory location (e.g., **8** or **Joe**).

Material from Dr. Ralph Morelli, Trinity College

Binary, Hex, and Decimal Numbers

This activity addresses the concept of abstraction. It focuses, in part, on the following learning objectives:

- 5b. Explanation of how number bases, including binary and decimal, are used for reasoning about digital data.

Introduction

As we've discussed, binary sequences are an important way to represent digital data. Our number system, which we call the *decimal* system, is a *base-10* number system. It uses 10 digits – 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 – and the places in a decimal number are based on powers of 10 – i.e., the ones place (10^0), tens place (10^1), hundreds place (10^2), and so on.

The *binary* system is a *base-2* system. It uses just 2 digits – 0 and 1 – and the places in the number are based on the powers of two. So we have the ones place (2^0), the twos place (2^1), the fours place (2^2), and so on.

Because binary numbers can get very long, computer scientists use other number systems based on powers of 2 that make it easier to represent digital data. One of these is the *hexadecimal* system, which is a *base-16* system. It has 16 digits – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. And the places in a hexadecimal number are based on powers of 16. So we have the ones place (16^0), the sixteens place, (16^1), the 256-place (16^2) and so on.

In this homework you will watch a short (10 minute) video about binary numbers and learn how to convert.

Khan Academy video on binary

- Binary numbers (<http://www.khanacademy.org/video/binary-numbers?playlist=Pre-algebra>)

Number Conversions

- Read about Number Conversions (<http://www.cstutoringcenter.com/tutorials/general/convert.php>) from CS Tutoring Center

Quiz Yourself

To reinforce your understanding of these concepts, try the following quizzes. If you can get ten-in-a-row correct, that's a pretty good indication that you get it.

- Binary to Decimal (<http://www.cs.trincoll.edu/~ram/q/110/binary-to-decimal.html>)
- Decimal to Binary (<http://www.cs.trincoll.edu/~ram/q/110/decimal-to-binary.html>)
- Hexadecimal to Decimal (<http://www.cs.trincoll.edu/~ram/q/110/hex-to-decimal.html>)
- Decimal to Hexadecimal (<http://www.cs.trincoll.edu/~ram/q/110/decimal-to-hex.html>)
- Binary to Hexadecimal (<http://www.cs.trincoll.edu/~ram/q/110/binary-to-hex.html>)
- Hexadecimal to Binary (<http://www.cs.trincoll.edu/~ram/q/110/hex-to-binary.html>)

Material from Dr. Ralph Morelli, Trinity College

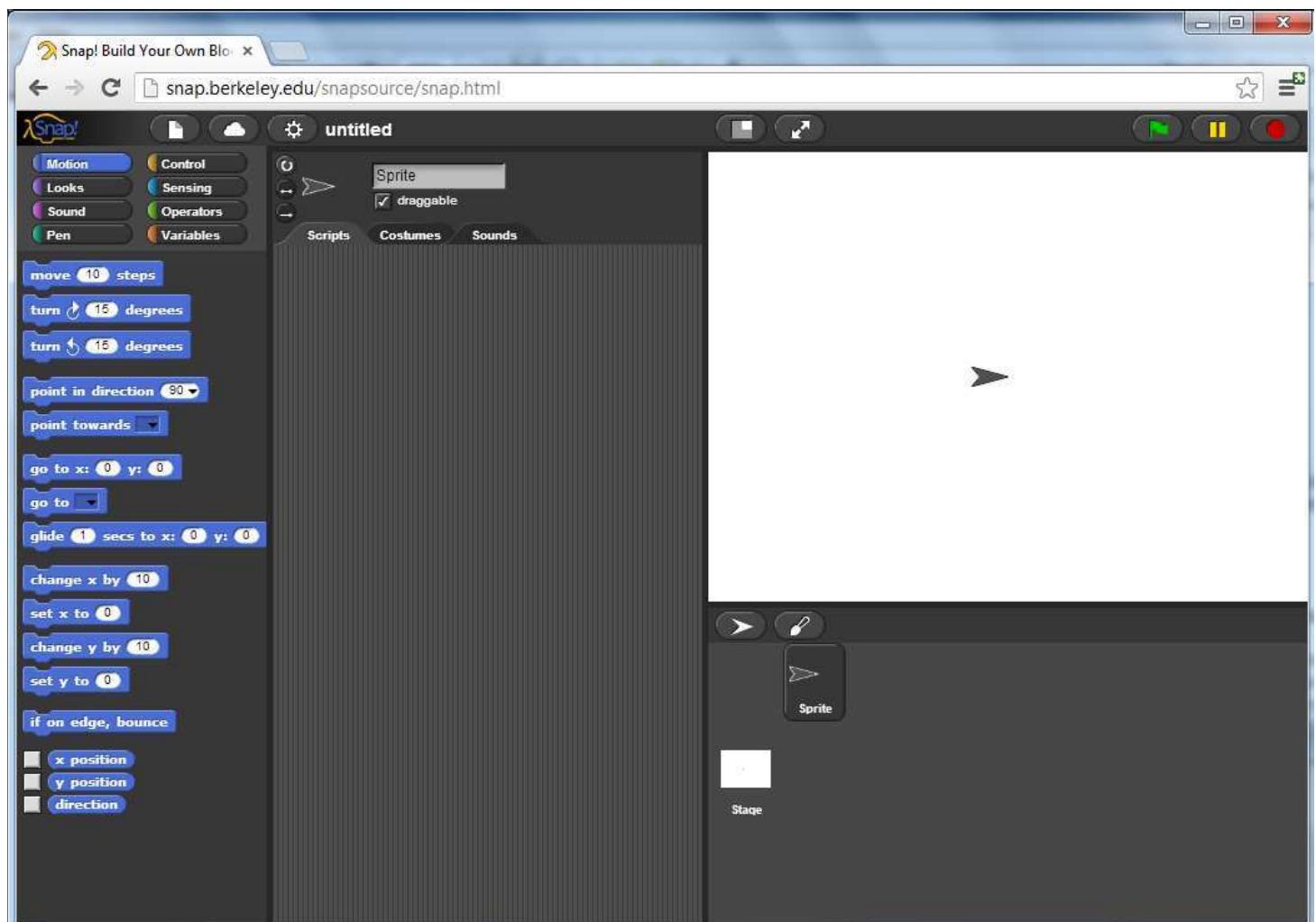
Conversion Exercise

1. Convert the following numbers to decimal notation.
 1. The binary number 111.
 2. The binary number 1011.
 3. The binary number 1011 1011.
 4. The hex number 61.
 5. The hex number DA.
 6. The hex number FEE.
 2. Convert the following decimal numbers as indicated.
 1. Convert decimal 12 to binary.
 2. Convert decimal 44 to binary.
 3. Convert decimal 254 to hex.
 4. Convert decimal 16 to hex.
 3. Challenge: Convert decimal 125 to octal (base 8) notation.
-

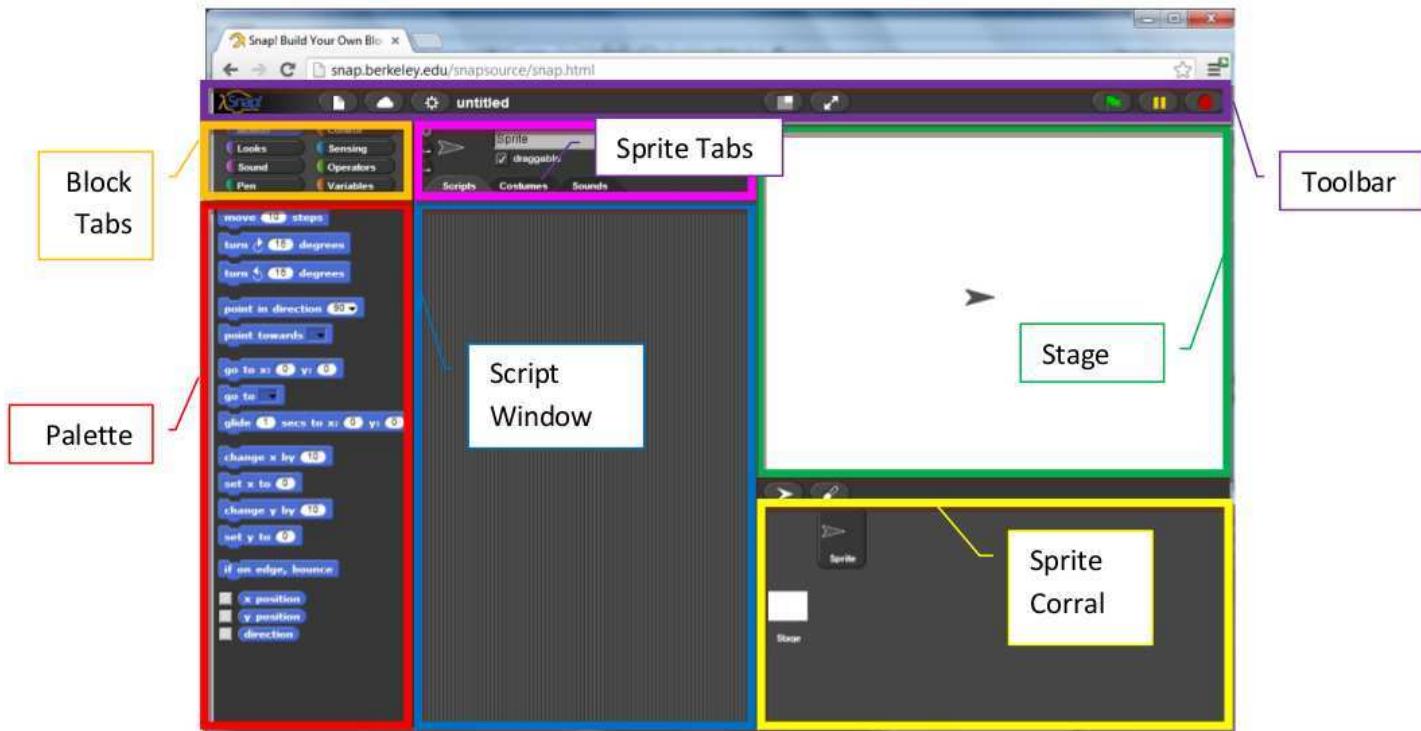
Lab: Welcome to Visual Programming

Let's open up SNAP at <http://snap.berkeley.edu/run> (<http://snap.berkeley.edu/run>)

You will see a screen like the one shown below. Explore the aspects of the user interface. Play around for a while and see if you can figure out the major components of the interface. In the next step, you will make your first project and explore further.



Let's first look at the IDE (Integrated Development Environment).



Make a sprite sing

For your first project, make a quick song! You will find the following **blocks** in the Sound tab useful; feel free to change the default numbers as you see fit.

While you are working on it, try to figure out how to connect and disconnect blocks, and how to remove a piece from inside a long script. Also, what do you think is the difference between these two blocks?

Hint: Try to use many copies of one of the blocks in a row, and hear the result. Do this for each block.

Meowing: One at a Time or in Unison?

With this brief introduction to the scratch interface, we begin to examine how sprites and blocks interact and affect one another. For example, the “play sound” blocks from earlier allow us to control when and how many sounds we hear. Consider the difference between these two blocks?



If you set up a small script like this, how many meows do you hear?





How about one like this: How many meows do you hear now?



Experiment with these blocks: 1) How about two “play sound (meow)” and then one “play sound (meow) until done”?



2) How about two “play sound (meow) until done” and then one “play sound (meow)”

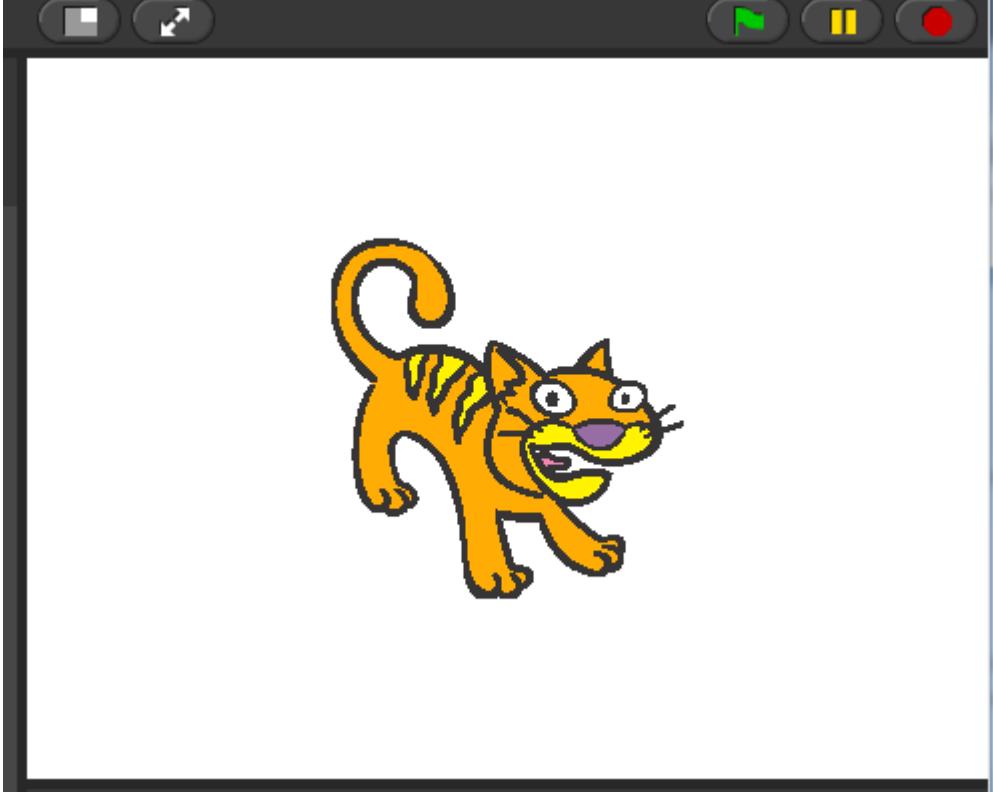


Explain the difference between 1) and 2). Why did you hear a different amount of meows?

Some Starting Lingo

Term	Example/Description
Tabs (for blocks)	
Tabs (for sprite)	
Blocks	
Script	

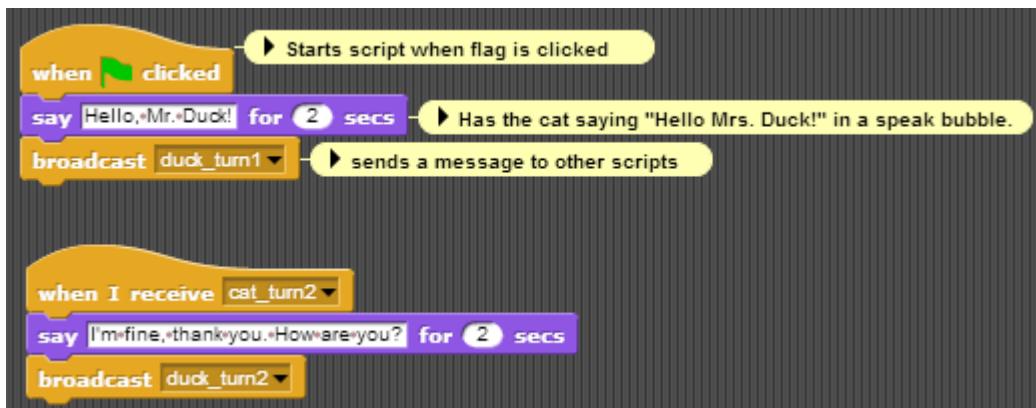
Term	Example/Description
Sprite	
Costumes (Each sprite can have multiple costumes)	 <p>Costumes (Each sprite can have multiple costumes)</p> <p>import a picture from another web page or from a file on your computer by dropping it here</p> <p>cat3</p> <p>cat4</p>

Term	Example/Description
Stage	
Bug	A defect (or a problem) in the code or routine of a program.
User Interface	The place where interaction between humans and machines occurs. For example, Windows, Scratch, the iPod touch screen, and even your keyboard are examples of user interfaces.

Experiment with a Short Play

Try to make these scripts in SNAP! You will find that the Cat and the Duck have completely separate script areas. Click on each character to see their script area. Once you are done, press the green flag to start the short play.

Cat's Script Window



Duck's Script Window



A note about style

You will notice that we chose to name the messages that were broadcast so that it would help us keep track of what we were doing and what messages we were sending. We recommend that you do this in your projects!

Hints

To choose a new sprite, drag an image file into the Costume area. Click on the paint brush to create a new image of your own.

Try to figure out what the commands (whose images are on the left) and buttons (whose images are on the right) do. These will be helpful to get the characters to face each other.

Try It! Play

Once you have this working, change the sprites, and then change the script of the "play" so that each character says at least two additional lines.

Exporting Sprites

By this point, you have probably figured out how to save your projects, but you can also save individual Sprites separately. To save (or export) a Sprite, right-click on the sprite and select export this sprite. To load (or import) a Sprite, click on the icon with a folder next to New Sprite (circled in yellow in the image below) and select the Sprite that you want to add to your project.

Curriculum (/bjc-course/curriculum) / Unit 1 (/bjc-course/curriculum/01-welcome) /
Lab 3 (/bjc-course/curriculum/01-welcome/labs/03-lights-camera-action) /

Lights, Camera, Action

In this activity, you will create a movie or a play.

In general, your movie should have at least:

- One block that we did not use during lab. (Explore your blocks)
- Two characters not previously used.
- Ten (10) total broadcasts of messages. This can be five each, or six/four, etc.
- Add sounds (Sound Tab)

The activity is open to your creativity – tell a story, create characters, experiment with moving sprites around stage

Unit 2: Loops and Variables

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts
- 3: The student can use computing tools and techniques for creative expression.
- 4: The student can use programming as a creative tool.
- 9: The student can use models and simulations to raise and answer questions.
- 15: The student can develop an algorithm.
- 16: The student can express an algorithm in a language.
- 28: The student can analyze how computing affects communication, interaction, and cognition.
- 29: The student can connect computing with innovations in other fields.
- 30: The student can analyze the beneficial and harmful effects of computing.
- 31: The student can connect computing within economic, social, and cultural contexts.

Readings/Lectures

- Blown to Bits: Chapter 2 (<http://www.bitsbook.com/wp-content/uploads/2008/12/chapter2.pdf>)
- Lecture Slides 2.01: Applications that changed the world (/bjc-course/curriculum/02-loops-and-variables/readings/01-applications-that-changed-the-world-slides.pdf)
- Lecture Slides 2.02: Algorithms (/bjc-course/curriculum/02-loops-and-variables/readings/02-algorithms-slides.pdf)
- Lecture Slides 2.03: Algorithm Development Activity (/bjc-course/curriculum/02-loops-and-variables/readings/03-algorithms-activity-slides.pdf)
- Lecture Slides 2.04: Loops and Variables (/bjc-course/curriculum/02-loops-and-variables/readings/04-loops-variables-slides.pdf)
- Lecture Video 2.05: Variables (/bjc-course/curriculum/02-loops-and-variables/readings/05-looping-byob-video.mp4)
- Lecture Video 2.06: Looping (/bjc-course/curriculum/02-loops-and-variables/readings/06-variables-byob-video.mp4)

Labs/Exercises

- Lab 2.01: Variables and Looping (/bjc-course/curriculum/02-loops-and-variables/labs/01-variables-and-looping)
- Lab 2.02: Position on the Stage (/bjc-course/curriculum/02-loops-and-variables/labs/02-position-on-the-stage)
- Lab 2.03: Repeat Until Practice Worksheet (/bjc-course/curriculum/02-loops-and-variables/labs/03-practice-repeat-until.pdf)
- Lab 2.04: Regular Figures Worksheet (/bjc-course/curriculum/02-loops-and-variables/labs/04-regular-figures-activity.pdf)

Why We Lost Our Privacy, or Gave It Away

Information technology did not cause the end of privacy, any more than automotive technology caused teen sex. Technology creates opportunities and risks, and people, as individuals and as societies, decide how to live in the changed landscape of new possibilities. To understand why we have less privacy today than in the past, we must look not just at the gadgets. To be sure, we should be wary of spies and thieves, but we should also look at those who protect us and help us—and we should also take a good look in the mirror.

We are most conscious of our personal information winding up in the hands of strangers when we think about data loss or theft. Reports like the one about the British tax office have become fairly common. The theft of information about 45 million customers of TJX stores, described in Chapter 5, “Secret Bits,” was even larger than the British catastrophe. In 2003, Scott Levine, owner of a mass email business named Snipermail, stole more than a billion personal information records from Acxiom. Millions of Americans are victimized by identity theft every year, at a total cost in the tens of billions of dollars annually. Many more of us harbor daily fears that just “a little bit” of our financial information has leaked out, and could be a personal time bomb if it falls into the wrong hands.

Why can’t we just keep our personal information to ourselves? Why do so many other people have it in the first place, so that there is an opportunity for it to go astray, and an incentive for creative crooks to try to steal it?

We lose control of our personal information because of things we do to ourselves, and things others do to us. Of things we do to be ahead of the curve, and things we do because everyone else is doing them. Of things we do to save money, and things we do to save time. Of things we do to be safe from our enemies, and things we do because we feel invulnerable. Our loss of privacy is a problem, but there is no one answer to it, because there is no one reason why it is happening. It is a messy problem, and we first have to think about it one piece at a time.

We give away information about ourselves—voluntarily leave visible footprints of our daily lives—because we judge, perhaps without thinking about it very much, that the benefits outweigh the costs. To be sure, the benefits are many.

Saving Time

For commuters who use toll roads or bridges, the risk-reward calculation is not even close. Time is money, and time spent waiting in a car is also anxiety and

frustration. If there is an option to get a toll booth transponder, many commuters will get one, even if the device costs a few dollars up front. Cruising past the cars waiting to pay with dollar bills is not just a relief; it actually brings the driver a certain satisfied glow.

The transponder, which the driver attaches to the windshield from inside the car, is an RFID, powered with a battery so identifying information can be sent to the sensor several feet away as the driver whizzes past. The sensor can be mounted in a constricted travel lane, where a toll booth for a human toll-taker might have been. Or it can be mounted on a boom above traffic, so the driver doesn't even need to change lanes or slow down.

And what is the possible harm? Of course, the state is recording the fact that the car has passed the sensor; that is how the proper account balance can be debited to pay the toll. When the balance gets too low, the driver's credit card may get billed automatically to replenish the balance. All that only makes the system better—no fumbling for change or doing anything else to pay for your travels.

The monthly bill—for the Massachusetts Fast Lane, for example—shows where and when you got on the highway—when, accurate to the second. It also shows where you got off and how far you went. Informing you of the mileage is another useful service, because Massachusetts drivers can get a refund on certain fuel taxes, if the fuel was used on the state toll road. Of course, you do not need a PhD to figure out that the state also knows when you got off the road, to the second, and that with one subtraction and one division, its computers could figure out if you were speeding. Technically, in fact, it would be trivial for the state to print the appropriate speeding fine at the bottom of the statement, and to bill your credit card for that amount at the same time as it was charging for tolls. That would be taking convenience a bit too far, and no state does it, yet.

What does happen right now, however, is that toll transponder records are introduced into divorce and child custody cases. You've never been within five miles of that lady's house? Really? Why have you gotten off the highway at the exit near it so many times? You say you can be the better custodial parent for your children, but the facts suggest otherwise. As one lawyer put it, "When a guy says, 'Oh, I'm home every day at five and I have dinner with my kids every single night,' you subpoena his E-ZPass and you find out he's crossing that bridge every night at 8:30. Oops!" These records can be subpoenaed, and have been, hundreds of times, in family law cases. They have also been used in employment cases, to prove that the car of a worker who said he was working was actually far from the workplace.

But most of us aren't planning to cheat on our spouses or our bosses, so the loss of privacy seems like no loss at all, at least compared to the time

saved. Of course, if we actually *were* cheating, we *would* be in a big hurry, and might take some risks to save a few minutes!

Saving Money

Sometimes it's money, not time, which motivates us to leave footprints. Such is the case with supermarket loyalty cards. If you do not want Safeway to keep track of the fact that you bought the 12-pack of Yodels despite your recent cholesterol results, you can make sure it doesn't know. You simply pay the "privacy tax"—the surcharge for customers not presenting a loyalty card. The purpose of loyalty cards is to enable merchants to track individual item purchases. (Item-level transactions are typically not tracked by credit card companies, which do not care if you bought Yodels instead of granola, so long as you pay the bill.) With loyalty cards, stores can capture details of cash transactions as well. They can process all the transaction data, and draw inferences about shoppers' habits. Then, if a lot of people who buy Yodels also buy Bison Brew Beer, the store's automated cash register can automatically spit out a discount coupon for Bison Brew as your Yodels are being bagged. A "discount" for you, and more sales for Safeway. Everybody wins. Don't they?

As grocery stores expand their web-based business, it is even easier for them to collect personal information about you. Reading the fine print when you sign up is a nuisance, but it is worth doing, so you understand what you are giving and what you are getting in return. Here are a few sentences of Safeway's privacy policy for customers who use its web site:

Safeway may use personal information to provide you with newsletters, articles, product or service alerts, new product or service announcements, saving awards, event invitations, personally tailored coupons, program and promotional information and offers, and other information, which may be provided to Safeway by other companies. ... We may provide personal information to our partners and suppliers for customer support services and processing of personal information on behalf of Safeway. We may also share personal information with our affiliate companies, or in the course of an actual or potential sale, re-organization, consolidation, merger, or amalgamation of our business or businesses.

Dreary reading, but the language gives Safeway lots of leeway. Maybe you don't care about getting the junk mail. Not everyone thinks it is junk, and the

company does let you “opt out” of receiving it (although in general, few people bother to exercise opt-out rights). But Safeway has lots of “affiliates,” and who knows how many companies with which it *might* be involved in a merger or sale of part of its business. Despite privacy concerns voiced by groups like C.A.S.P.I.A.N. (Consumers Against Supermarket Privacy Invasion and Numbering, www.nocards.org), most shoppers readily agree to have the data collected. The financial incentives are too hard to resist, and most consumers just don’t worry about marketers knowing their purchases. But whenever purchases can be linked to your name, there is a record, somewhere in a huge database, of whether you use regular or super tampons, lubricated or unlubricated condoms, and whether you like regular beer or lite. You have authorized the company to share it, and even if you hadn’t, the company could lose it accidentally, have it stolen, or have it subpoenaed.

Convenience of the Customer

The most obvious reason not to worry about giving information to a company is that you do business with them, and it is in your interest to see that they do their business with you better. You have no interest in whether they make more money from you, but you do have a strong interest in making it easier and faster for you to shop with them, and in cutting down the amount of stuff they may try to sell you that you would have no interest in buying. So your interests and theirs are, to a degree, aligned, not in opposition. Safeway’s privacy policy states this explicitly: “Safeway Club Card information and other information may be used to help make Safeway’s products, services, and programs more useful to its customers.” Fair enough.

No company has been more progressive in trying to sell customers what they might want than the online store Amazon. Amazon suggests products to repeat customers, based on what they have bought before—or what they have simply looked at during previous visits to Amazon’s web site. The algorithms are not perfect; Amazon’s computers are drawing inferences from data, not being clairvoyant. But Amazon’s guesses are pretty good, and recommending the wrong book every now and then is a very low-cost mistake. If Amazon does it too often, I might switch to Barnes and Noble, but there is no injury to me. So again: Why should anyone care that Amazon knows so much about me? On the surface, it seems benign. Of course, we don’t want the credit card information to go astray, but who cares about knowing what books I have looked at online?

Our indifference is another marker of the fact that we are living in an exposed world, and that it feels very different to live here. In 1988, when a

How SITES KNOW WHO YOU ARE

1. You tell them. Log in to Gmail, Amazon, or eBay, and you are letting them know exactly who you are.
2. They've left cookies on one of your previous visits. A *cookie* is a small text file stored on your local hard drive that contains information that a particular web site wants to have available during your current session (like your shopping cart), or from one session to the next. Cookies give sites persistent information for tracking and personalization. Your browser has a command for showing cookies—you may be surprised how many web sites have left them!
3. They have your IP address. The web server has to know where you are so that it can ship its web pages to you. Your IP address is a number like 66.82.9.88 that locates your computer in the Internet (see the Appendix for details). That address may change from one day to the next. But in a residential setting, your Internet Service Provider (your *ISP*—typically your phone or cable company) knows who was assigned each IP address at any time. Those records are often subpoenaed in court cases.

If you are curious about who is using a particular IP address, you can check the American Registry of Internet Numbers (www.arin.net). Services such as whatismyip.com, whatismyip.org, and ipchicken.com also allow you to check your own IP address. And www.whois.net allows you to check who owns a domain name such as harvard.com—which turns out to be the Harvard Bookstore, a privately owned bookstore right across the street from the university. Unfortunately, that information won't reveal who is sending you spam, since spammers routinely forge the source of email they send you.

videotape rental store clerk turned over Robert Bork's movie rental records to a Washington, DC newspaper during Bork's Supreme Court confirmation hearings, Congress was so outraged that it quickly passed a tough privacy protection bill, The Video Privacy Protection Act. Videotape stores, if any still exist, can be fined simply for keeping rental records too long. Twenty years later, few seem to care much what Amazon does with its millions upon millions of detailed, fine-grained views into the brains of all its customers.

It's Just Fun to Be Exposed

Sometimes, there can be no explanation for our willing surrender of our privacy except that we take joy in the very act of exposing ourselves to public

view. Exhibitionism is not a new phenomenon. Its practice today, as in the past, tends to be in the province of the young and the drunk, and those wishing to pretend they are one or the other. That correlation is by no means perfect, however. A university president had to apologize when an image of her threatening a Hispanic male with a stick leaked out from her MySpace page, with a caption indicating that she had to “beat off the Mexicans because they were constantly flirting with my daughter.”

And there is a continuum of outrageousness. The less wild of the party photo postings blend seamlessly with the more personal of the blogs, where the bloggers are chatting mostly about their personal feelings. Here there is

*Bits don't fade and they
don't yellow. Bits are forever.
And we don't know how to
live with that.*

not exuberance, but some simpler urge for human connectedness. That passion, too, is not new. What is new is that a photo or video or diary entry, once posted, is visible to the entire world, and that there is no taking it

back. Bits don't fade and they don't yellow. Bits are forever. And we don't know how to live with that.

For example, a blog selected with no great design begins:

This is the personal web site of Sarah McAuley. ... I think sharing my life with strangers is odd and narcissistic, which of course is why I'm addicted to it and have been doing it for several years now. Need more? You can read the “About Me” section, drop me an email, or you know, just read the drivel that I pour out on an almost-daily basis.

No thank you, but be our guest. Or consider that there is a Facebook group just for women who want to upload pictures of themselves uncontrollably drunk. Or the Jennicam, through which Jennifer Kay Ringley opened her life to the world for seven years, setting a standard for exposure that many since have surpassed in explicitness, but few have approached in its endless ordinariness. We are still experimenting, both the voyeurs and viewed.

Because You Can't Live Any Other Way

Finally, we give up data about ourselves because we don't have the time, patience, or single-mindedness about privacy that would be required to live our daily lives in another way. In the U.S., the number of credit, debit, and bank cards is in the billions. Every time one is used, an electronic handshake records a few bits of information about who is using it, when, where, and for what. It is now virtually unheard of for people to make large purchases of

ordinary consumer goods with cash. Personal checks are going the way of cassette tape drives, rendered irrelevant by newer technologies. Even if you could pay cash for everything you buy, the tax authorities would have you in their databases anyway. There even have been proposals to put RFIDs in currency notes, so that the movement of cash could be tracked.

Only sects such as the Amish still live without electricity. It will soon be almost that unusual to live without Internet connectivity, with all the finger-prints it leaves of your daily searches and logins and downloads. Even the old dumb TV is rapidly disappearing in favor of digital communications. Digital TV will bring the advantages of video on demand—no more trips to rent movies or waits for them to arrive in the mail—at a price: Your television service provider will record what movies you have ordered. It will be so attractive to be able to watch what we want when we want to watch it, that we won't miss either the inconvenience or the anonymity of the days when all the TV stations washed your house with their airwaves. You couldn't pick the broadcast times, but at least no one knew which waves you were grabbing out of the air.

Little Brother Is Watching

So far, we have discussed losses of privacy due to things for which we could, in principle anyway, blame ourselves. None of us really needs a loyalty card, we should always read the fine print when we rent a car, and so on. We would all be better off saying “no” a little more often to these privacy-busters, but few of us would choose to live the life of constant vigilance that such resolute denial would entail. And even if we were willing to make those sacrifices, there are plenty of other privacy problems caused by things others do to us.

The snoopy neighbor is a classic American stock figure—the busybody who watches how many liquor bottles are in your trash, or tries to figure out whose Mercedes is regularly parked in your driveway, or always seems to know whose children were disorderly last Saturday night. But in Cyberspace, we are all neighbors. We can all check up on each other, without even opening the curtains a crack.

Public Documents Become VERY Public

Some of the snooping is simply what anyone could have done in the past by paying a visit to the Town Hall. Details that were always public—but inaccessible—are quite accessible now.

suspected threats to defense facilities as part of a larger program of domestic counterintelligence.

The Transportation Security Administration (TSA) is responsible for airline passenger screening. One proposed system, CAPPS II, which was ultimately terminated over privacy concerns, sought to bring together disparate data sources to determine whether a particular individual might pose a transportation threat. Color-coded assessment tags would determine whether you could board quickly, be subject to further screening, or denied access to air travel.

The government creates projects, the media and civil liberties groups raise serious privacy concerns, the projects are cancelled, and new ones arise to take their place. The cycle seems to be endless. In spite of Americans' traditional suspicions about government surveillance of their private lives, the cycle seems to be almost an inevitable consequence of Americans' concerns about their security, and the responsibility that government officials feel to use the best available technologies to protect the nation. Corporate databases often contain the best information on the people about whom the government is curious.

Technology Change and Lifestyle Change

New technologies enable new kinds of social interactions. There were no suburban shopping malls before private automobiles became cheap and widely used. Thirty years ago, many people getting off an airplane reached for cigarettes; today, they reach for cell phones. As Heraclitus is reported to have said 2,500 years ago, "all is flux"—everything keeps changing. The reach-for-your-cell phone gesture may not last much longer, since airlines are starting to provide onboard cell phone coverage.

The more people use a new technology, the more useful it becomes. (This is called a "network effect"; see Chapter 4, "Needles in the Haystack.") When one of us got the email address `lewis@harvard` as a second-year graduate student, it was a vainglorious joke; all the people he knew who had email addresses were students in the same office with him. Email culture could not develop until a lot of people had email, but there wasn't much point in having email if no one else did.

Technology changes and social changes reinforce each other. Another way of looking at the technological reasons for our privacy loss is to recognize that the social institutions enabled by the technology are now more important than the practical uses for which the technology was originally conceived. Once a lifestyle change catches on, we don't even think about what it depends on.

Credit Card Culture

The usefulness of the data aggregated by Acxiom and its kindred data aggregation services rises as the number of people in their databases goes up, and as larger parts of their lives leave traces in those databases. When credit cards were mostly short-term loans taken out for large purchases, the credit card data was mostly useful for determining your creditworthiness. It is still useful for that, but now that many people buy virtually everything with credit cards, from new cars to fast-food hamburgers, the credit card transaction database can be mined for a detailed image of our lifestyles. The information is there, for example, to determine if you usually eat dinner out, how much traveling you do, and how much liquor you tend to consume. Credit card companies do in fact analyze this sort of information, and we are glad they do. If you don't seem to have been outside Montana in your entire life and you turn up buying a diamond bracelet in Rio de Janeiro, the credit card company's computer notices the deviation from the norm, and someone may call to be sure it is really you.

The credit card culture is an economic problem for many Americans, who accept more credit card offers than they need, and accumulate more debt than they should. But it is hard to imagine the end of the little plastic cards, unless even smaller RFID tags replace them. Many people carry almost no cash today, and with every easy swipe, a few more bits go into the databases.

Email Culture

Email is culturally in between telephoning and writing a letter. It is quick, like telephoning (and instant messaging is even quicker). It is permanent, like a letter. And like a letter, it waits for the recipient to read it. Email has, to a great extent, replaced both of the other media for person-to-person communication, because it has advantages of both. But it has the problems that other communication methods have, and some new ones of its own.

Phone calls are not intended to last forever, or to be copied and redistributed to dozens of other people, or to turn up in court cases. When we use email as though it were a telephone, we tend to forget about what else might happen to it, other than the telephone-style use, that the recipient will read it and throw it away. Even Bill Gates probably wishes that he had written his corporate emails in a less telephonic voice. After testifying in an antitrust lawsuit that he had not contemplated cutting a deal to divide the web browser market with a competitor, the government produced a candid email he had sent, seeming to contradict his denial: "We could even pay them money as part of the deal, buying a piece of them or something."

Email is as public as postcards, unless it is encrypted, which it usually is not.

appropriate advertising. If you are working within a financial services corporation, your emails are probably logged—even the ones to your grandmother—because the company has to be able to go back and do a thorough audit if something inappropriate happens.

Email is as public as postcards, unless it is encrypted, which it usually is not. Employers typically reserve the right to read what is sent through company email. Check the policy of your own employer; it may be hard to find, and it may not say what you expect. Here is Harvard's policy, for example:

Employees must have no expectation or right of privacy in anything they create, store, send, or receive on Harvard's computers, networks, or telecommunications systems. Electronic files, e-mail, data files, images, software, and voice mail may be accessed at any time by management or by other authorized personnel for any business purpose. Access may be requested and arranged through the system(s) user, however, this is not required.

Employers have good reason to retain such sweeping rights; they have to be able to investigate wrongdoing for which the employer would be liable. As a result, such policies are often less important than the good judgment and ethics of those who administer them. Happily, Harvard's are generally good. But as a general principle, the more people who have the authority to snoop, the more likely it is that someone will succumb to the temptation.

Commercial email sites can retain copies of messages even after they have been deleted. And yet, there is very broad acceptance of public, free, email services such as Google's Gmail, Yahoo! Mail, or Microsoft's Hotmail. The technology is readily available to make email private: whether you use encryption tools, or secure email services such as Hushmail, a free, web-based email service that incorporates PGP-based encryption (see Chapter 5). The usage of these services, though, is an insignificant fraction of their unencrypted counterparts. Google gives us free, reliable email service and we, in return, give up some space on our computer screen for ads. Convenience and cost trump privacy. By and large, users don't worry that Google, or its competitors, have all their mail. It's a bit like letting the post office keep a copy of every letter you send, but we are so used to it, we don't even think about it.

Email is bits, traveling within an ISP and through the Internet, using email software that may keep copies, filter it for spam, or submit it to any other form of inspection the ISP may choose. If your email service provider is Google,

the point of the inspection is to attach some

Web Culture

When we send an email, we think at least a *little* bit about the impression we are making, because we are sending it to a human being. We may well say things we would not say face-to-face, and live to regret that. Because we can't see anyone's eyes or hear anyone's voice, we are more likely to over-react and be hurtful, angry, or just too smart for our own good. But because email is directed, we don't send email thinking that no one else will ever read what we say.

The Web is different. Its social sites inherit their communication culture not from the letter or telephone call, but from the wall in the public square, littered with broadsides and scribbled notes, some of them signed and some not. Type a comment on a blog, or post a photo on a photo album, and your action can be as anonymous as you wish it to be—you do not know to whom your message is going. YouTube has millions of personal videos. Photo-archiving sites are the shoeboxes and photo albums of the twenty-first century. Online backup now provides easy access to permanent storage for the contents of our personal computers. We entrust commercial entities with much of our most private information, without apparent concern. The generation that has grown up with the Web has embraced social networking in all its varied forms: MySpace, YouTube, LiveJournal, Facebook, Xanga, Classmates.com, Flickr, dozens more, and blogs of every shape and size. More than being taken, personal privacy has been given away quite freely, because everyone else is doing it—the surrender of privacy is more than a way to social connectedness, it is a social institution in its own right. There are 70 million bloggers sharing everything from mindless blather to intimate personal details. Sites like www.loopt.com let you find your friends, while twitter.com lets you tell the entire world where you are and what you are doing. The Web is a confused, disorganized, chaotic realm, rich in both gold and garbage.

The “old” web, “Web 1.0,” as we now refer to it, was just an information resource. You asked to see something, and you got to see it. Part of the disinhibition that happens on the new “Web 2.0” social networking sites is due to the fact that they still allow the movie-screen illusion—that we are “just looking,” or if we are contributing, we are not leaving footprints or fingerprints if we use pseudonyms. (See Chapter 4 for more on Web 1.0 and Web 2.0.)

But of course, that is not really the way the Web ever worked. It is important to remember that even Web 1.0 was never anonymous, and even “just looking” leaves fingerprints.

In July 2006, a *New York Times* reporter called Thelma Arnold of Lilburn, Georgia. Thelma wasn't expecting the call. She wasn't famous, nor was she involved in anything particularly noteworthy. She enjoyed her hobbies, helped her friends, and from time to time looked up things on the Web—stuff about her dogs, and her friends' ailments.

Then AOL, the search engine she used, decided to release some “anonymous” query data. Thelma, like most Internet users, may not have known that AOL had kept every single topic that she, and every other one of their users, had asked about. But it did. In a moment of unenlightened generosity, AOL released for research use a small sample: about 20 million queries from 658,000 different users. That is actually not a lot of data by today's standards. For example, in July 2007, there were about 5.6 billion search engine queries, of which roughly 340 million were AOL queries. So, 20 million queries comprise only a couple of days' worth of search queries. In an effort to protect their clients' privacy, AOL “de-identified” the queries. AOL never mentioned anyone by name; they used random numbers instead. Thelma was 4417149. AOL mistakenly presumed that removing a single piece of personal identification would make it hard to figure out who the users were. It turned out that for some of the users, it wasn't hard at all.

It didn't take much effort to match Thelma with her queries. She had searched for “landscapers in Lilburn, GA” and several people with the last name “Arnold,” leading to the obvious question of whether there were any Arnolds in Lilburn. Many of Thelma's queries were not particularly useful for identifying her, but were revealing nonetheless: “dry mouth,” “thyroid,” “dogs that urinate on everything,” and “swing sets.”

Thelma was not the only person to be identified. User 22690686 (Terri) likes astrology, and the Edison National Bank, Primerica, and Budweiser. 5779844 (Lawanna) was interested in credit reports, and schools. From what he searched for, user 356693 seems to have been an aide to Chris Shays, Congressman from Connecticut.

One of the privacy challenges that we confront as we rummage through the rubble of the digital explosion is that information exists without context. Was Thelma Arnold suffering from a wide range of ailments? One might readily conclude that from her searches. The fact is that she often tried to help her friends by understanding their medical problems.

Or consider AOL user 17556639, whose search history was released along with Thelma Arnold's. He searched for the following:

how to kill your wife 23 Mar, 22:09
wife killer 23 Mar, 22:11
poop 23 Mar, 22:12
dead people 23 Mar, 22:13
pictures of dead people 23 Mar, 22:15
killed people 23 Mar, 22:16
dead pictures 23 Mar, 22:17
murder photo 23 Mar, 22:20
steak and cheese 23 Mar, 22:22
photo of death 23 Mar, 22:30
death 23 Mar, 22:33
dead people photos 23 Mar, 22:33
photo of dead people 23 Mar, 22:35
www.murderedpeople.com 23 Mar, 22:37
decapitated photos 23 Mar, 22:39
car crashes3 23 Mar, 22:40
car crash photo 23 Mar, 22:41

Is this AOL user a potential criminal? Should AOL have called the police? Is 17556639 about to kill his wife? Is he (or she) a researcher with a spelling problem and an interest in Philly cheese steak? Is reporting him to the police doing a public service, or is it an invasion of privacy?

There is no way to tell just from these queries if this user was contemplating some heinous act or doing research for a novel that involves some grisly scenes. When information is incomplete and decontextualized, it is hard to judge meaning and intent.

In this particular case, we happen to know the answer. The user, Jason from New Jersey, was just fooling around, trying to see if Big Brother was watching. He wasn't planning to kill his wife at all. Inference from incomplete data has the problem of false positives—thinking you have something that you don't, because there are other patterns that fit the same data.

Information without context often leads to erroneous conclusions. Because our digital trails are so often retrieved outside the context within which they were created, they sometimes suggest incorrect interpretations. Data interpretation comes with balanced social responsibilities, to protect society when there is evidence of criminal behavior or intent, and also to protect the individual when such evidence is too limited to be reliable. Of course, for every example of misleading and ambiguous data, someone will want to solve the problems it creates by collecting more data, rather than less.

Beyond Privacy

There is nothing new under the sun, and the struggles to define and enforce privacy are no exception. Yet history shows that our concept of privacy has evolved, and the law has evolved with it. With the digital explosion, we have arrived at a moment where further evolution will have to take place rather quickly.

Leave Me Alone

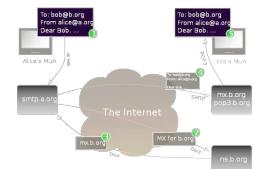
More than a century ago, two lawyers raised the alarm about the impact technology and the media were having on personal privacy:

Instantaneous photographs and newspaper enterprise have invaded the sacred precincts of private and domestic life; and numerous mechanical devices threaten to make good the prediction that “what is whispered in the closet shall be proclaimed from the house-tops.”

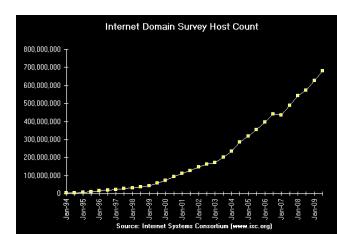
This statement is from the seminal law review article on privacy, published in 1890 by Boston attorney Samuel Warren and his law partner, Louis Brandeis, later to be a justice of the U.S. Supreme Court. Warren and Brandeis went on, “Gossip is no longer the resource of the idle and of the vicious, but has become a trade, which is pursued with industry as well as effrontery. To satisfy a prurient taste the details of sexual relations are spread broadcast in the columns of the daily papers. To occupy the indolent, column upon column is filled with idle gossip, which can only be procured by intrusion upon the domestic circle.” New technologies made this garbage easy to produce, and then “the supply creates the demand.”

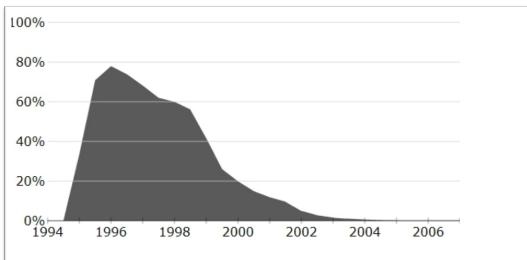
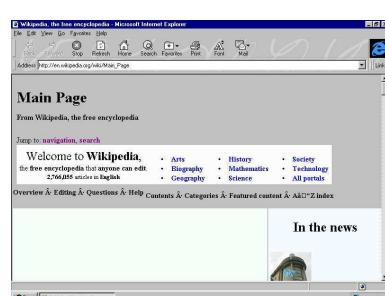
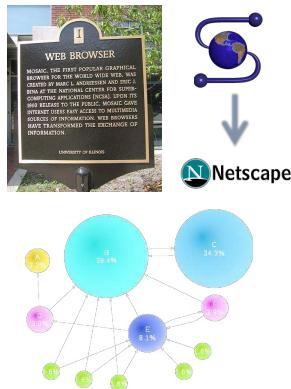
And those candid photographs and gossip columns were not merely tasteless; they were bad. Sounding like modern critics of mindless reality TV, Warren and Brandeis raged that society was going to hell in a handbasket because of all that stuff that was being spread about.

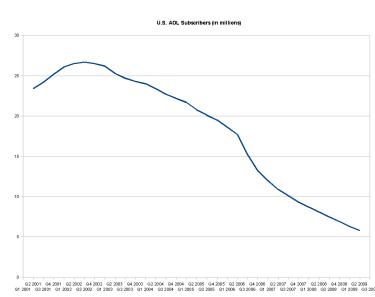
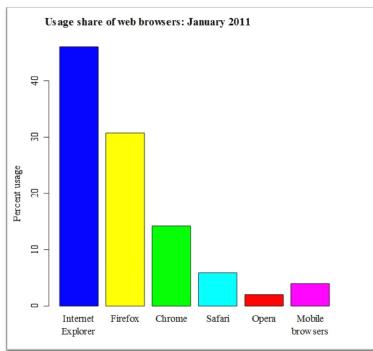
Even gossip apparently harmless, when widely and persistently circulated, is potent for evil. It both belittles and perverts. It belittles by inverting the relative importance of things, thus dwarfing the thoughts and aspirations of a people. When personal gossip attains the dignity of print, and crowds the space available for matters of



en.wikipedia.org/wiki/Personal_computer





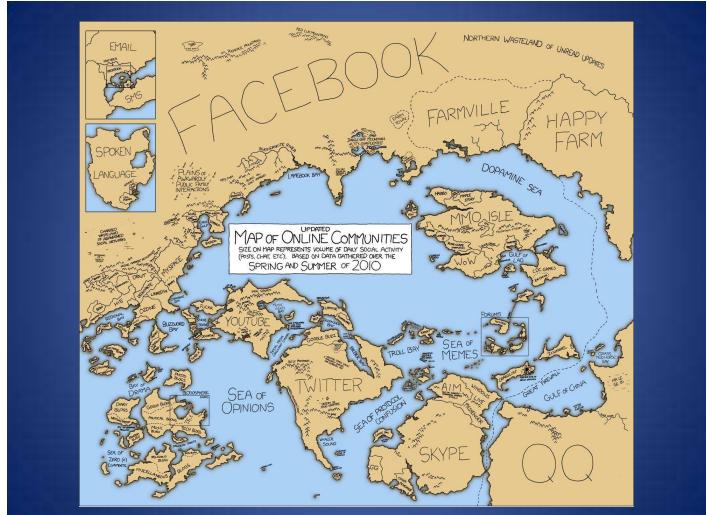
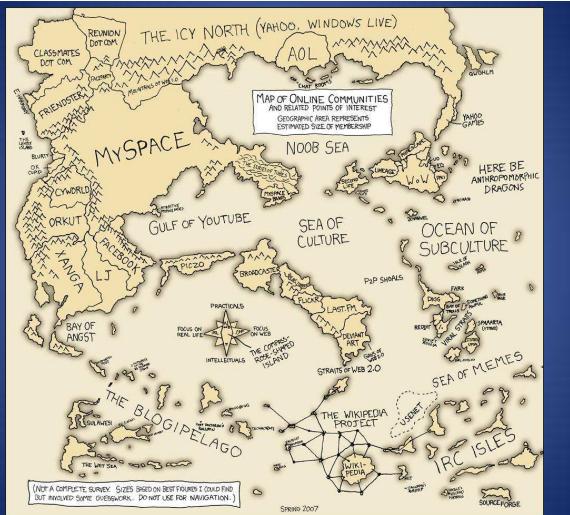




facebook

A screenshot of the original Thefacebook website. It shows a blue header with the text "[thefacebook]". Below the header is a login form with fields for "Email:" and "Password:", and buttons for "register" and "login". A message on the page says "Welcome to Thefacebook! We have opened up Thefacebook for popular consumption at Harvard University." It lists ways to use the site, including searching for people and looking up friends' friends. At the bottom, there are "Register" and "Login" buttons, along with links for "about", "contact", "FAQ", "terms", and "privacy". A copyright notice at the bottom states "a Mark Zuckerberg production Thefacebook © 2004".

facebook



Facebook Growth

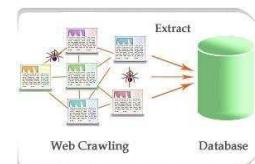
- <http://www.penn-olson.com/2010/02/10/infographic-facebook-s-amazing-growth/>

Facebook's Impacts

- How has Facebook affected you?
 - Personal Life
 - Academic / Professional Life
- How has Facebook affected the world?
 - Literally, has revolutionized countries and overthrown governments.

Facebook

- What “shoulders” are they standing on?



Google docs





<http://www.fastcompany.com/1733627/mit-scientist-captures-his-sons-first-90000-hours-on-video>



Algorithms

Computer Science Principles

Problem Solving Tools

- Programs are created to solve problems.
- A solution must be designed prior to coding.
- One method of designing a solution to a problem is to create an **algorithm**.

Algorithms

- An **algorithm** is a list of steps to solve a problem written in plain English.
 - **Steps** to solve a problem are written out and numbered in the order in which they should be executed.
- They should be as extensive as necessary to outline the solution.
- Your algorithm is not only going to tell your program what to do but how to do it.

Algorithm Example – Going Home

The Walk Algorithm

1. Leave classroom
2. Turn right out of school building
3. Walk 1.2 miles
4. Turn right on street
5. Go to 4th house

The Bus Algorithm

1. Go to the bus area
2. Get in right bus
3. Go to house

Algorithms

- Both algorithms, and others that accomplish the same task (of getting you home).
- There are advantages and disadvantages associated with each option.
- You have to consider each option and its advantages/disadvantages before you choose the algorithm you want to continue developing into your program.

Programming Algorithm Example

Simple steps representing a process for dealing with a guessing game in which the computer generates a random number and the player guesses.

1. Generate a secret random number between 1 and 100.
2. Get a number from the player.
3. Compare the player's guess to the secret number.
4. Compare the numbers. If the numbers are identical, go to step 5. Otherwise, tell the player the number was either too high and return to step 2.
5. Display a message stating the secret number was guessed.

Pseudocode

- **Pseudocode** is a mix of English language and code that represents what you want your program to do.
- It helps you determine how you want the program to work as well as what variables and methods/functions you will want to include.
- Developing pseudocode will help you work through your logic, reducing the number of errors and potential re-writes you will have to do.

Pseudocode Example

Represents the same process for dealing with a guessing game in which the computer generates a random number and the player guesses the number

```
btnCheckGuess_Click()
randomNumber = 37
Get playerGuess from text box
If playerGuess = randomNumber Then
    Display "Correct"
Elseif playerGuess < randomNumber Then
    Display "Guess too Low"
Else
    Display "Guess too High"
End
```

Flowchart

- A third tool in programming is through the use of a **flowchart**.
- Flowcharts use symbols and text to give a visual representation of a solution to a problem.
- The direction of the arrows indicates the flow of the logic.

Flowchart

- Flowcharts help the programmer begin to plan the programming project.
- They provide a **visual representation** of the algorithm or process.
- They describe the inputs, processes and outputs of the program that are needed to successfully complete the project.

Flowchart Symbols

- There are many flowchart programs, however you can also use Microsoft Word to create a flowchart – or just a piece of paper and a pencil.
- To create the flowchart, there are different symbols that represent the various parts. We will only use a few of these symbols.
- Use lines with arrows to indicate flow of control.
- The text in your flowchart symbols is your pseudocode.

Flowchart Symbols

Start/End

Input/Output

Processes

Decisions

Ovals : Start should always be the first shape, with an End at the end of the flow chart or a process.

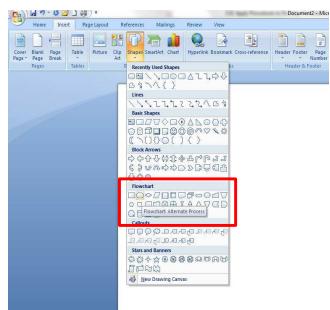
Parallelogram: This shape is used to show raw materials used for 'ingredients' and to show the finished product. Input/Output – Get/Display

Rectangles should be used to show processes/commands, eg. 'Bake Cake'. These are activities.

Diamonds: Hold questions that resolve into True or False. Used for **decisions** that divide into two options and to control loops.

Using Microsoft Word to Create a Flowchart

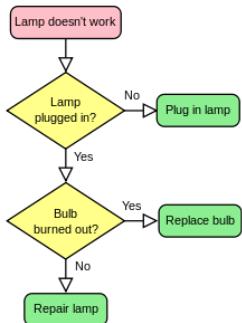
- Open Microsoft Word.
- Under Insert choose Shapes
- Look down the list until you see Flowchart.
- Hover your mouse over a shape, you will see a popup telling you what that shape is used for.



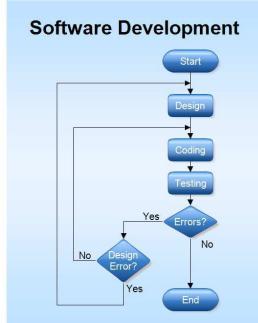
Using Microsoft Word to Create a Flowchart

- Select and draw the shapes needed for your program logic.
- Once you draw a shape you can right click and select Add Text to enter information into your symbol.
- Join your symbols using arrows indicating program data flow.

Flowchart Examples



<http://en.wikipedia.org/wiki/Flowchart>



http://www.rff.com/software_development.htm

Expectations

- Some general rules:
 - If you do not understand the problem, you probably will not be able to create a solution.
 - Remember to start with the solution in mind.
 - Your program solution should not necessarily look like that of another programmer.
 - Use your tools to help you determine your solution.

Algorithm Development

Computer Science Inside...

What written instructions have you followed...?

- ...to complete a task?
 - Can you give an example?
- Were the instructions easy or difficult to follow?
 - Why? What made them easy/ hard?
 - They made sense?
 - You couldn't understand them?
 - They didn't give you enough information?

Following an Algorithm

- Algorithm written on hand-out
 - to draw a picture
- You cannot ask for any help
- Don't look at your classmates work
 - do it by yourself!!!

What is the similarity between these?

- Cooking recipe
- Downloading software or music
- Car repair manual
- Setting up a music playlist
- Knitting pattern
- Calling a friend on the phone
- Sheet music



Why discuss lists of instructions here??

- Computer programs are lists of instructions
 - with very particular characteristics
 - known as *algorithms*
- How many of you know of a famous computer error/mistake?
- These are caused by the wrong *instructions* in the program
 - the instructions were interpreted by the computer in a way not intended by the program designer
- We are going to explore how these errors come about



Here's the algorithm – follow exactly!!

1. Draw a diagonal line
2. Draw another diagonal line connected to the top of the first one
3. Draw a straight line from the point where the diagonal lines meet
4. Draw a horizontal line over the straight line
5. At the bottom of the straight line, draw a curvy line
6. Draw a diagonal line from the bottom of the first diagonal to the straight line
7. Draw a diagonal line from the bottom of the second diagonal to the straight line



How did the pictures turn out?

RESULTS

- Compare your picture with others' pictures...
 - Were they different?
 - Why?
 - What was difficult about following the instructions
 - What was missing from the instructions?

- Let's look at your results.





Putting all this together...

- This time:
 - write an Algorithm
 - test it yourself
 - get someone else to try it out...
- Can you be sure your algorithm will work ok?





Following an algorithm

- Hide your shape
- Get into pairs
 - by teaming up with someone **on the opposite side of the room**
 - move to sit together
 - **Do not** show them your paper shape – hide it!!
- Swap algorithm/instructions with your partner
- Follow your partner's instructions to create their paper shape
- Compare shapes
 - how similar is each 'pair' of shapes?
 - what advice can you give on how to improve the instructions?





Write & test your algorithm

- The task/problem:
 - make a shape out of paper – one sheet of A4
- Write the *algorithm*
 - Write a set of instructions that explains how to make a paper shape from 1 sheet of A4 paper
- Test it
 - Try out your algorithm – does it work?
 - Note: follow your instructions **as closely as possible**
 - Adjust the instructions if necessary



What do we know about algorithms?

- What are the key characteristics of a “good” algorithm? Why are they hard to develop?
 - Must be unambiguous
 - Must be correct
 - Must be at the right level of detail
- Also, what did we learn about problems we pick?
 - too large sometimes?





Algorithms are fundamental...

- ...to Computer Science, and to society
 - Our electronic devices are teeming with algorithms realised in programming code
 - You perform them every day, every hour...

- First algorithms developed by the Greeks
 - e.g. Euclidean algorithm for finding greatest common divisor
- "Algorithm" comes from Al Khwarizmi – Persian astronomer and mathematician

CS Inside...



Some activities are not *algorithmic* in nature

- Problem solving
- Human thinking process
- Falling in love
- and so on...
- That is why some might think of these are *hard*...!!
- When we can express the human thinking process as an algorithm, Artificial Intelligence will have truly been created

CS Inside...

Conclusions

- Algorithm
 - step-by-step method for accomplishing a task
- Following an algorithm
 - relatively easy
- Finding/designing Algorithms
 - difficult but exciting and fulfilling
 - the designed algorithm contains the intelligence of its developer
- Algorithms are a fundamental part of Computer Programming and of Computing Science



Loops & Variables

Computer Science Principles

VARIABLES

What is a Variable?

- A variable is a named space in memory.
- Think of a mailroom with a large wall of slots for the mail as your memory.
 - This is very simplified of course.
- Each of these slots would be assigned to a variable (by its name) and would hold the values assigned.



Adding Variables

- You can add variables to your program to increase its flexibility.
- The variable allows you to change a value as the script runs.
- To add a variable, select the Variables tab, then click on the Make a Variable button.



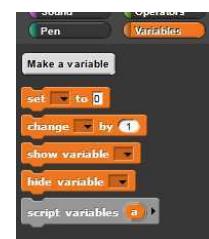
Creating a Variable

- When you click on the Make a Variable button, the Variable Name window will open.
- Note you can create the variable for the active sprite only (called a local variable) or for all sprites (called a global variable).



Variable Blocks

- You have multiple variable blocks.
- Checking the checkbox beside a variable name will display the value of the variable on the stage. (only visible when there is a variable.)
- **set [variableName] to ()**
 - Sets the value
- **change [variableName] by ()**
 - Changes the value
- **show variable [variableName]**
 - Displays the value on the stage.
- **hide variable [variableName]**
 - Displays the value on the stage.
- **script variables (a)**
 - Creates local variables



Example of Variable Use

- Create a variable called mynote that will be the value of what note is played.
 - Now the note will change as the loop runs (from using the `repeat ()` block).
 - Note that we had to give the variable a starting value.
 - This is called initializing the variable.



Set vs. Change

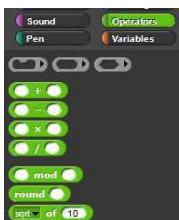
- Note that using a `set [variableName] to ()` block will set the value of the variable – NOT update it.
- To update or change a value, use the `change [variableName] by ()` block.



° LOOPING

Helping Blocks

- There are blocks that you will want to use with your variables and loops.
- These blocks are in the Operator's palette.



Looping

- There are times when we want certain blocks to repeat more than one time.
- There are blocks that allow us to do just that.
 - `warp`
 - Does not show the interim steps – only the final product
 - `forever`
 - Will continue to loop until the program closes
 - This is basically an infinite loop as it goes on forever.
 - `repeat ()`
 - Will continue to loop the specified number of times.
 - `repeat until < >`
 - Will continue to loop until the condition is met (true)



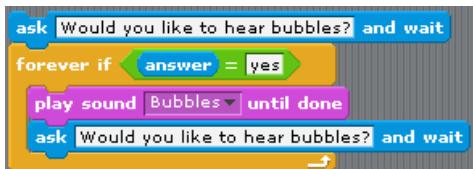
Looping Blocks

- Will continue to play the Bubbles sound.
- Will play the Bubbles sound three times



Looping Example

- Will ask the question, then wait for the answer.
- If the answer is "yes" it will play the Bubbles sound.
- Then ask the question again and wait for the answer.
- Playing and asking the question will continue to loop until the answer is something other than "yes"



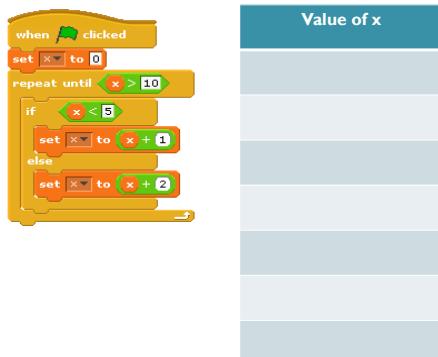
Looping Example

- Let's look at the "repeat until" block a bit closer.
- Just like REPEAT, it will do everything inside the C-shaped block a certain number of times.
- However before it starts the loop each time, it checks to see if the condition ($x > 5$) is true.
- When this condition is true, it will not repeat again.



Before the loop	0
Top of loop	0
Bottom of loop	1
Top of loop	1
Bottom of loop	2
Top of loop	2
Bottom of loop	3
Top of loop	3
Bottom of loop	4
Top of loop	4
Bottom of loop	5
Top of loop	5
Bottom of loop	6

repeat until <>



DRAWING BLOCKS

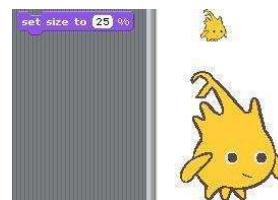
Important Blocks for Drawing

- There are several blocks you will use to draw.
 - move () steps**
 - Will move your sprite which will draw for you.
 - turn () degrees**
 - Will turn your sprite to face that direction
 - clear**
 - Will clear your stage
 - pen down**
 - Will tell the sprite to start drawing
 - pen up**
 - Will tell the sprite to stop drawing



Changing Sprite Size

- In order to see your drawing, you might want to change the size of your sprite.
- In the Looks area, you will set the **set size to () %** block.

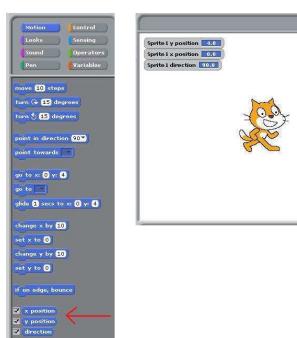


Where is my Sprite?

- You might also need to know where your sprite is located by the x and y positions as well as the direction your sprite is facing on the stage.

- Look in the Motion area, you will see the several blocks you can use.

- By checking these blocks, the information will be displayed on the stage.

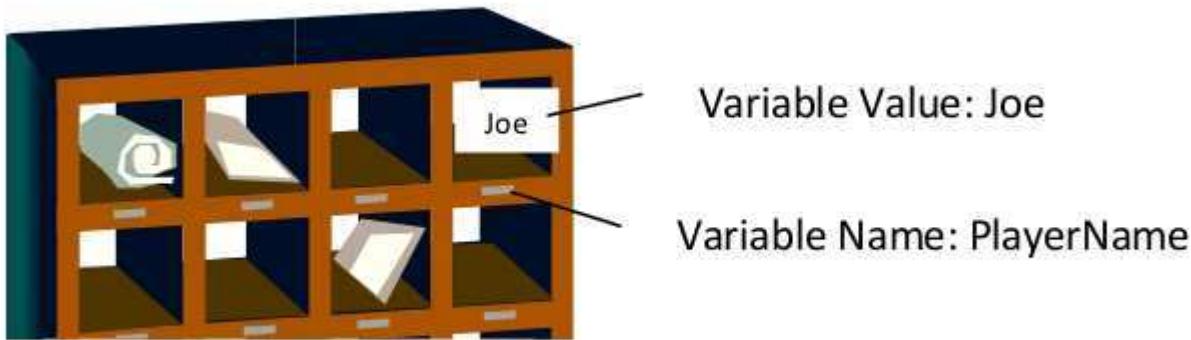


Lab: Variables and looping

Let's open up SNAP at <http://snap.berkeley.edu/run> (<http://snap.berkeley.edu/run>)

Variables

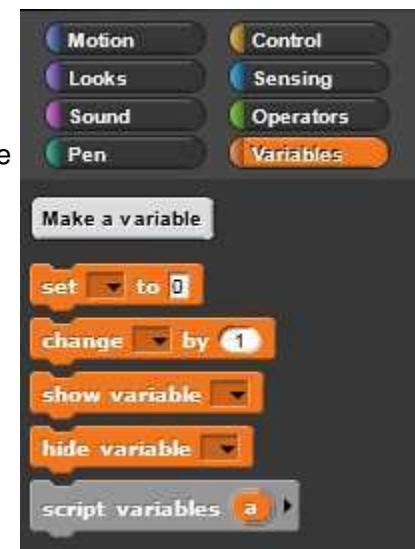
Variables are named spaces in memory that your program can access. You can set and modify the values that are contained in these named spaces. To visualize a variable's name space think of mail slots in a large mail room. Your computer has memory that the program is going to use to create and store information. When you create a variable, you are assigning one of the "slots" of memory to a name and then can put a value in that slot and modify it as needed.



Why create variables? Variables allow the programmer to make the value modifiable in the script. For example, you want to be able to update a score variable as the player wins/loses in a game. You will see many different uses of variables during this course.

You have multiple blocks to create and manipulate variables in the Variables tab palette.

- **Make a variable** button allows you to create a new variable
- **Delete a variable** button will allow you to delete a button – this button only shows after you have created a variable.
- **Set [] to (0)** will allow you to initialize, or set a beginning value, the variable to a value.
- **Change [] by (1)** allows you to modify the value of a variable
- **Show variable []** will show the variable and value on the stage.
- **hide variable []** will hide the variable and value on the stage.
- **Script variables (a)** will allow you to create local variables, more on this use later



To create a variable

- In the box that pops up - type the name of the variable. The default selection "for all sprites" means that all sprites have access to this variable. Select "for this sprite only" if you want only the sprite currently selected to be able to access/modify the value of the variable.



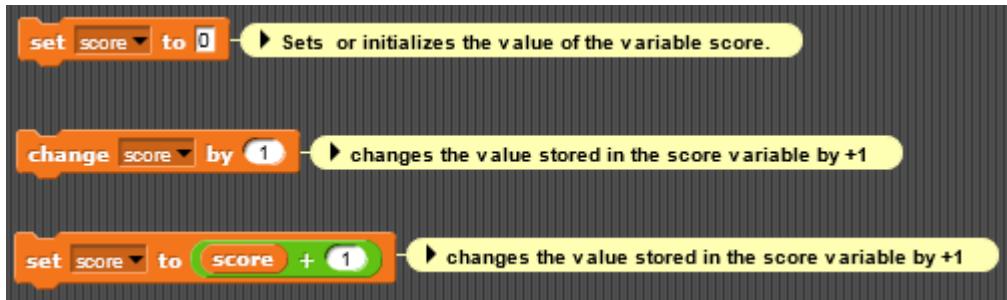
- Now you'll have blocks to use for your variable.
- Note that you now have a rounded button with the name of the variable in the window with a checkbox to the left. If checked, the variable and value will be show on the stage. Uncheck to hide.
- When you use one of the variable blocks you will be able to click on the combo box arrow for a list of your variables.



Common Bug: Set vs. Change

A very common mistake is to use a change block when you need a set block or use a set block when you need a change block.

It is actually a little more complicated, in that you can make your set block act like a change block.

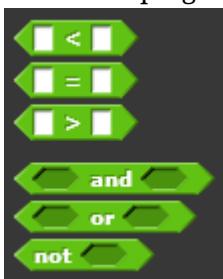


Looping Blocks

There are times when you will want blocks to repeat. Instead of duplicating blocks and ending up with a long script that might be confusing, there are looping control blocks that you can wrap around the script you want to repeat.

- `forever` Loops until the program ends. This is basically an *infinite loop* as it goes on forever.
- `repeat ()` Loops the specified number of times.
- `repeat until < >` Repeat until the condition is *True*.

For the `repeat until < >` you will use a predicate block that returns true or false. These blocks have pointed ends and can be found in the Operators palette.



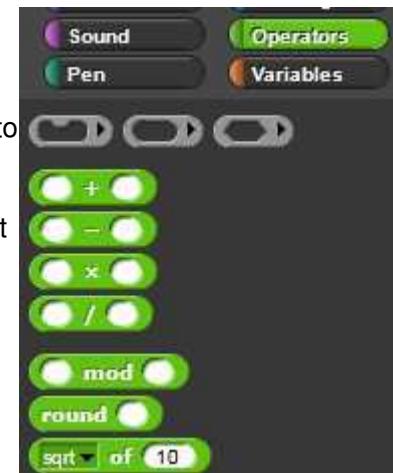
Other helpful blocks include the operator blocks.

Operators

Click the Operators tab to display a new palette of blocks. You can use these blocks to perform mathematic operations to modify a numeric variable.

You have blocks to add, subtract, multiple and divide. You also have a mod block that does remainder division as well as a round block and a square root block.

NOTE: To see what any block does, right click on a block in the palette and select help. A new window will open that will explain what that block does.



Looping Examples



Plays the sound continuously until the program is stopped.

Plays the sound completely 10 times then stops

Plays the sound until the `score` variable's value is equal to 10

Repeat Until

Let's look at the "repeat until" block a bit closer. Just like REPEAT, it will do everything inside the C-shaped block a certain number of times. However before it starts the loop each time, it checks to see if the condition ($x > 5$) is true. When this condition is true, it will not repeat again.

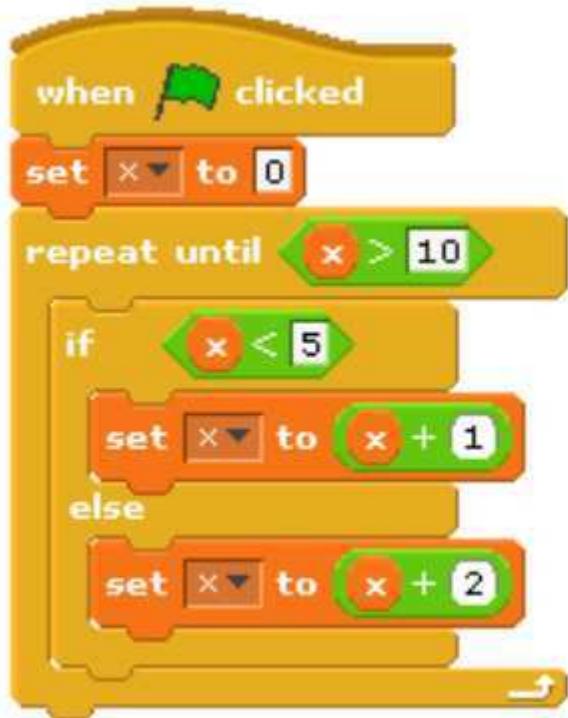


It can be really helpful to keep track of what the variable X is at each point to help us understand how this new piece

- In the right column we keep track of the value of x.
- In the diagram below we draw a horizontal line every time we start the loop. Here we labeled each line “Top of loop” and “Bottom of loop” but we could just use the horizontal line to keep track of this information.
- Within each loop the variable x increases by 1, so we write down the new value for x.

	
Before the loop	0
Top of loop	0
Bottom of loop	1
Top of loop	1
Bottom of loop	2
Top of loop	2
Bottom of loop	3
Top of loop	3
Bottom of loop	4
Top of loop	4
Bottom of loop	5
Top of loop	5
Bottom of loop	6

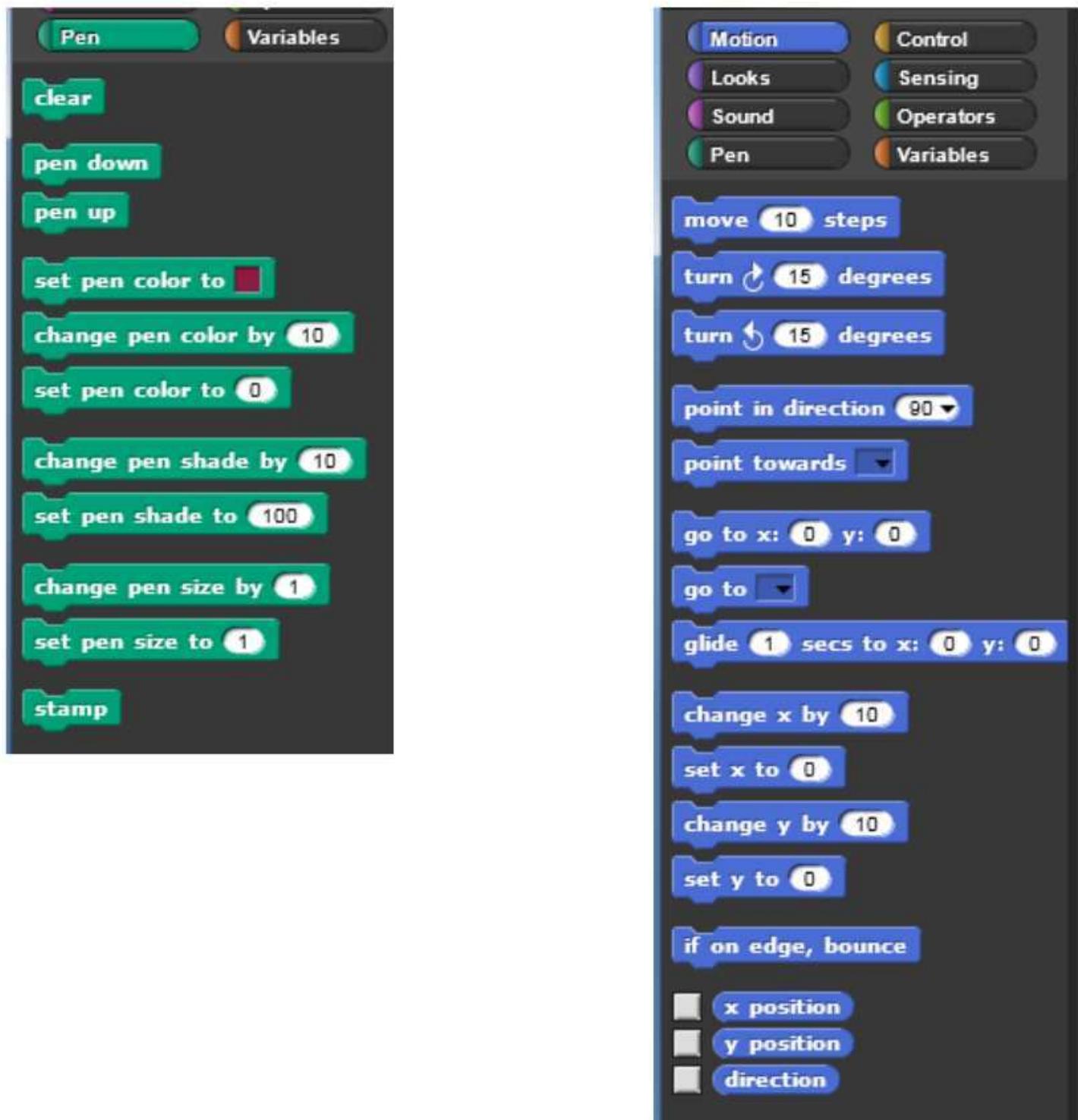
Try to use a chart like the one above to keep track of what happens in the complicated “repeat until” code below.



Value of x

Drawing Tools

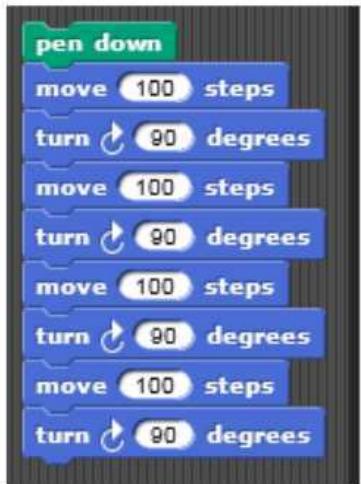
There are multiple blocks that can be used to draw. By combining blocks from the Pen palette with blocks from the Motion palette, you can draw pictures. Your sprite needs to face in the direction you want the line to be drawn so you will need the point in direction () block.



Try it! Drawing shapes

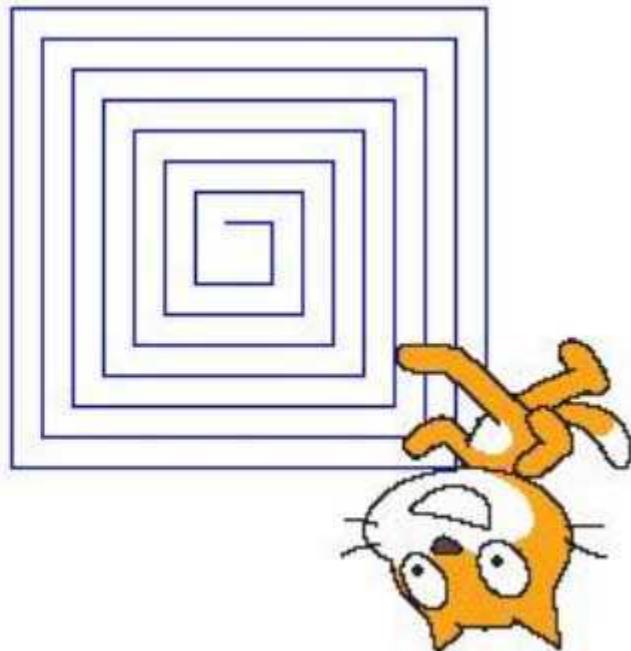
Look at the following scripts to draw a square. The first script has repetitive code. In the second script, the repeating code has been replaced by using a loop.

Note: By using the `pen down` block your sprite will draw for you.



Exercise: Draw a Squaril

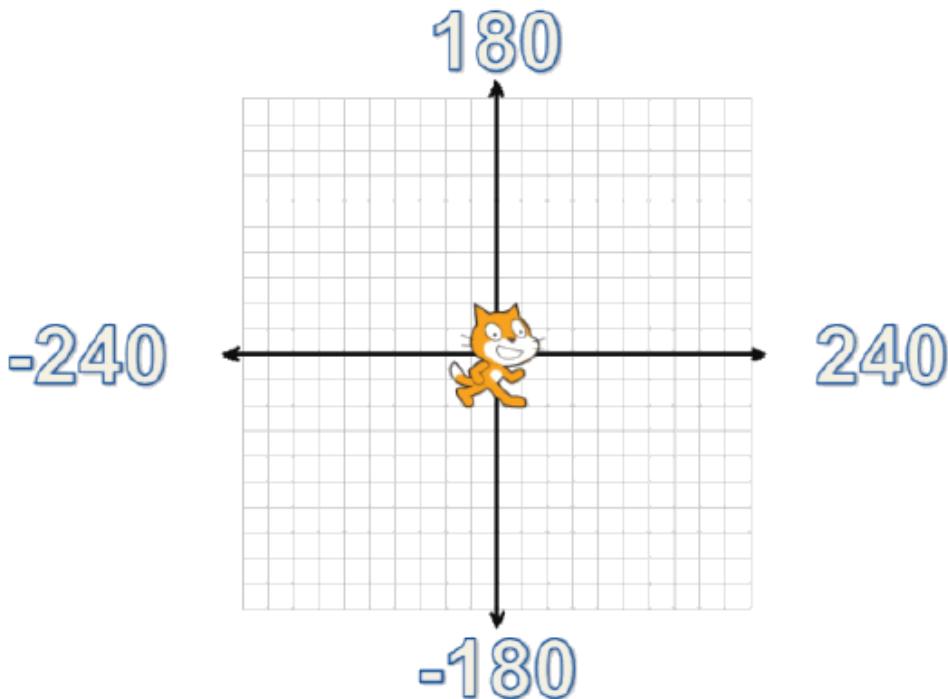
Make a new variable Length with a looping structure to draw the “Squiral” (Square + Spiral) below. Note that the length the sprite moves is updating by a constant amount.



Exercise: Position on the stage

The sprite occupies a point(x,y) on the stage corresponding to the x- and y- axis. Here's a picture:

Position on the Stage!



In Scratch, the sprite usually begins at the center of the stage. Its position is **(0,0)**.

The next lesson asks you to move a random character around the stage within certain boundaries. Keep in mind the coordinate system shown above.

Computer Science Principles

Practice Repeat Until

Name: _____

Partner Name: _____

Use the script block to evaluate the value of x.

Example



Iterations	x
1	1
2	2
3	3
4	4
5	5
6	7
7	9
8	11

Script in Pseudocode

When Flag Clicked

Set x to 0

Repeat until $x > 10$

if $x < 5$

set x to x + 1

else

set x to x + 2

End Loop

What is the value of x after the repeat block is finished? 11

1. Try this



b. How many times did this iterate: _____

c. How many times did you enter the "if" block: _____

2. Try this



b. How many times did this iterate: _____

3. Not so Crazy

a) Make up a not too crazy repeat until block.

b) Switch with a classmate and have them evaluate your script.

c) Draw the script here: (You can use Pseudocode)

d) Evaluate it here:

Iterations	x

4. Evil repeat-until

- a) Make up an evil repeat until block.
- b) Switch with a classmate and have them evaluate your script.
- c) Draw the script here: (You can use Pseudocode)

- d) Evaluate it here:

Iterations	x

- e) How many times did this iterate: _____

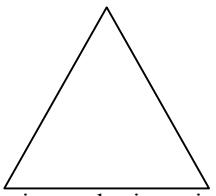
Name: _____

Beauty and Joy of Computing: Drawing Regular Figures

In Geometry, you learned (or will learn) how to calculate the exterior angle of any regular-polygon. In this lesson you will draw a series of regular figures. We want to take a second to make sure understand the math behind this.

We'll take it one figure at a time:

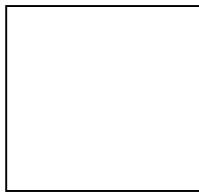
Step 1. The Triangle:



- a. Sum of the interior angles in a triangle: 180°
- b. Measure of each interior angle : _____
- c. Measure of each exterior angle: _____ <== This is the key!
- d. Sum of the exterior angles : _____

Step 2: Scratch It Now draw the triangle.

Step 3: The Square:



- a. Measure of each interior angle: _____
- b. Measure of each exterior angle: _____
- c. Sum of the exterior angles: _____

Step 4: Scratch It ... Now draw the square

Step 5: Putting it together:

- a. What do you notice about the sum of the exterior angles in the triangle and square:
- b. How is the sum of the exterior angles, the number of sides, and the measure of each exterior angle related:

Step 6: Extend to any figure ...

. Given what you figured out in Step 5, finish the chart

# sides	Sum of Exterior Angles	Measure of exterior angle
3		
4		
5		
6		
7		
8		
9		
10		
1000		
n		

Step 7: Scratch It ... Draw the figures above

Step 8: The five-sided star is tougher ... think about what regular-polygon would circumscribe the five-pointed star.

Unit 3: Loops and Variables

Procedural Abstraction

Learning Objectives

- 4: The student can use programming as a creative tool.
- 20: The student can use abstractions (procedures) to manage complexity in a program.

Readings/Lectures

- Blown to Bits: Chapter 3 (<http://www.bitsbook.com/wp-content/uploads/2008/12/chapter3.pdf>)
- Lecture Slides 3.01: Functions (/bjc-course/curriculum/03-build-your-own-blocks/readings/01-functions-slides.pdf)
- Lecture Video 3.02: Functions (/bjc-course/curriculum/03-build-your-own-blocks/readings/02-functions-video.mp4)

Labs/Exercises

- Lab 3.01: Build your own blocks (/bjc-course/curriculum/03-build-your-own-blocks/labs/01-build-your-own-blocks)
 - Lab 3.02: Project: Brick Wall (/bjc-course/curriculum/03-build-your-own-blocks/labs/02-brick-wall-project)
-

Who uses steganography today, if anyone? It is very hard to know. *USA Today* reported that terrorists were communicating using steganography in early 2001. A number of software tools are freely available that make steganography easy. Steganographic detectors—what are properly known as steganalysis tools—have also been developed, but their usefulness as yet seems to be limited. Both steganography and steganalysis software is freely available on the World Wide Web (see, for example, www.cotse.com/tools/stega.htm and www.outguess.org/detection.php).

The use of steganography to transmit secret messages is today easy, cheap, and all but undetectable. A foreign agent who wanted to communicate with parties abroad might well encode a bit string in the tonal values of an MP3 or the color values of pixels in a pornographic image on a web page. So much music and pornography flows between the U.S. and foreign countries that the uploads and downloads would arouse no suspicion!

The Scary Secrets of Old Disks

By now, you may be tempted to delete all the files on your disk drive and throw it away, rather than run the risk that the files contain unknown secrets. That isn't the solution: Even deleted files hold secrets!

A few years ago, two MIT researchers bought 158 used disk drives, mostly from eBay, and recovered what data they could. Most of those who put the disks up for sale had made some effort to scrub the data. They had dragged files into the desktop trash can. Some had gone so far as to use the Microsoft Windows FORMAT command, which warns that it will destroy all data on the disk.

Yet only 12 of the 158 disk drives had truly been sanitized. Using several methods well within the technical capabilities of today's teenagers, the researchers were able to recover user data from most of the others. From 42 of the disks, they retrieved what appeared to be credit card numbers. One of the drives seemed to have come from an Illinois automatic teller machine and contained 2,868 bank account numbers and account balances. Such data from single business computers would be a treasure trove for criminals. But most of the drives from home computers also contained information that the owners would consider extremely sensitive: love letters, pornography, complaints about a child's cancer therapy, and grievances about pay disputes, for example. Many of the disks contained enough data to identify the primary user of the computer, so that the sensitive information could be tied back to an individual whom the researchers could contact.

CLOUD COMPUTING

One way to avoid having problems with deleted disk files and expensive document-processing software is not to keep your files on your disks in the first place! In "cloud computing," the documents stay on the disks of a central service provider and are accessed through a web browser. "Google Docs" is one such service, which boasts very low software costs, but other major software companies are rumored to be exploring the market for cloud computing. If Google holds your documents, they are accessible from anywhere the Internet reaches, and you never have to worry about losing them—Google's backup procedures are better than yours could ever be. But there are potential disadvantages. Google's lawyers would decide whether to resist subpoenas. Federal investigators could inspect bits passing through the U.S., even on a trip between other countries.

the entire document may be physically scattered about the disk, although logically it is held together as a chain of references of one block to another. Logically, the structure is that of a magazine, where articles do not necessarily occupy contiguous pages. Part of an article may end with "Continued on page 152," and the part of the article on page 152 may indicate the page on which it is continued from there, and so on.

Because the files on a disk begin at random places on disk, an *index* records which files begin where on the disk. The index is itself another disk file, but one whose location on the disk can be found quickly. A disk index is very much like the index of a book—which always appears at the end, so readers know where to look for it. Having found the index, they can quickly find the page number of any item listed in the index and flip to that page.

The users of the computers had for the most part done what they thought they were supposed to do—they deleted their files or formatted their disks. They probably knew not to release toxic chemicals by dumping their old machines in a landfill, but they did not realize that by dumping them on eBay, they might be releasing personal information into the digital environment. Anyone in the world could have bought the old disks for a few dollars, and all the data they contained. What is going on here, and is there anything to do about it?

Disks are divided into blocks, which are like the pages of a book—each has an identifying address, like a page number, and is able to hold a few hundred bytes of data, about the same amount as a page of text in a book. If a document is larger than one disk block, however, the document is typically not stored in consecutive disk blocks. Instead, each block includes a piece of the document, and the address of the block where the document is continued. So

Why aren't disks themselves organized like books, with documents laid out on consecutive blocks? Because disks are different from books in two important respects. First, they are dynamic. The information on disks is constantly being altered, augmented, and removed. A disk is less like a book than like a three-ring binder, to which pages are regularly added and removed as information is gathered and discarded. Second, disks are perfectly re-writable. A disk block may contain one string of 0s and 1s at one moment, and as a result of a single writing operation, a different string of 0s and 1s a moment later. Once a 0 or a 1 has been written in a particular position on the disk, there is no way to tell whether the bit previously in that position was a 0 or a 1. There is nothing analogous to the faint traces of pencil marks on paper that are left after an erasure. In fact, there is no notion of "erasure" at all on a disk—all that ever happens is replacement of some bits by others.

Because disks are dynamic, there are many advantages to breaking the file into chained, noncontiguous blocks indexed in this way. For example, if the file contains a long text document and a user adds a few words to the middle of the text, only one or two blocks in the middle of the chain are affected. If enough text is added that those blocks must be replaced by five new ones, the new blocks can be logically threaded into the chain without altering any of the other blocks comprising the document. Similarly, if a section of text is deleted, the chain can be altered to "jump over" the blocks containing the deleted text.

Blocks that are no longer part of any file are added to a "pool" of available disk blocks. The computer's software keeps track of all the blocks in the pool. A block can wind up in the pool either because it has never been used or because it has been used but abandoned. A block may be abandoned because the entire file of which it was part has been deleted or because the file has been altered to exclude the block. When a fresh disk block is needed for any purpose—for example, to start a new file or to add to an existing file—a block is drawn from the pool of available blocks.

What Happens to the Data in Deleted Files?

Disk blocks are not re-written when they are abandoned and added to the pool. When the block is withdrawn from the pool and put back to work as part of another file, it is overwritten and the old data is obliterated. But until then, the block retains its old pattern of zeroes and ones. The entire disk file may be intact—except that there is no easy way to find it. A look in the index will reveal nothing. But "deleting" a file in this way merely removes the index entry. The information is still there on the disk somewhere. It has no more

been eradicated than the information in a book would be expunged by tearing out the index from the back of the volume. To find something in a book without an index, you just have to go through the book one page at a time looking for it—tedious and time-consuming, but not impossible.

And that is essentially what the MIT researchers did with the disks they bought off eBay—they went through the blocks, one at a time, looking for recognizable bit patterns. A sequence of sixteen ASCII character codes representing decimal digits, for example, looks suspiciously like a credit card number. Even if they were unable to recover an entire file, because some of the blocks comprising it had already been recycled, they could recognize significant short character strings such as account numbers.

Of course, there would be a simple way to prevent sensitive information from being preserved in fragments of “deleted” files. The computer could be programmed so that, instead of simply putting abandoned blocks into the

pool, it actually over-wrote the blocks, perhaps by “zeroing” them—that is, writing a pattern of all `0s`. Historically, computer and software manufacturers have thought the benefits of zeroing blocks far less than the costs. Society has not found “data leakage” to be a critical problem until recently—although that may be changing. And the costs of constantly zeroing disk blocks would be significant. Filling blocks with zeroes might take so much time that the users would complain about how slowly their machines were running

if every block were zeroed immediately. With some clever programming the process could be made unnoticeable, but so far neither Microsoft nor Apple has made the necessary software investment.

And who has not deleted a file and then immediately wished to recover it? Happily for all of us who have mistakenly dragged the wrong file into the trash can, as computers work today, deleted files are not immediately added to the pool—they can be dragged back out. Files can be removed only until you execute an “Empty trash” command, which puts the deleted blocks into the pool, although it does not zero them.

But what about the Windows “FORMAT” command, shown in Figure 3.16? It takes about 20 minutes to complete. Apparently it is destroying all the bits on the disk, as the warning message implies. But that is not what is happen-

THE LAW ADJUSTS

Awareness is increasing that deleted data can be recovered from disks. The Federal Trade Commission now requires “the destruction or erasure of electronic media containing consumer information so that the information cannot practicably be read or reconstructed,” and a similar provision is in a 2007 Massachusetts Law about security breaches.

ing. It is simply looking for faulty spots on the disk. Physical flaws in the magnetic surface can make individual disk blocks unusable, even though mechanically the disk is fine and most of the surface is flawless as well. The FORMAT command attempts to *read* every disk block in order to identify blocks that need to be avoided in the future. Reading every block takes a long time, but rewriting them all would take twice as long. The FORMAT command identifies the bad blocks and re-initializes the index, but leaves most of the data unaltered, ready to be recovered by an academic researcher—or an inventive snooper.

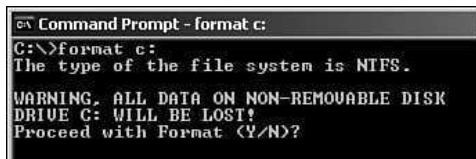


FIGURE 3.16 Warning screen of Microsoft Windows FORMAT command. The statement that all the data will be lost is misleading—in fact, a great deal of it can be recovered.

As if the problems with disks were not troubling enough, exactly the same problems afflict the memory of cell phones. When people get rid of their old phones, they forget the call logs and email messages they contain. And if they do remember to delete them, using the awkward combinations of button-pushes described deep in the phone's documentation, they may not really have accomplished what they hoped. A researcher bought ten cell phones on eBay and recovered bank account numbers and passwords, corporate strategy plans, and an email exchange between a woman and her married boyfriend, whose wife was getting suspicious. Some of this information was recovered from phones whose previous owners had scrupulously followed the manufacturer's instructions for clearing the memory.

SOFTWARE TO SCRUB YOUR DISK

If you really want to get rid of all the data on your disk, a special "Secure empty trash" command is available on Macintosh computers. On Windows machines, DBAN is free software that really will zero your disk, available through dban.sourceforge.net, which has lots of other useful free software. Don't use DBAN on your disk until you are sure you don't want anything on it anymore!

In a global sense, bits turn out to be very hard to eradicate. And most of the time, that is exactly the way we want it. If our computer dies, we are glad that Google has copies of our data. When our cell phone dies, we are happy if our contact lists reappear, magically downloaded from our cellular service provider to our replacement phone. There are upsides and downsides to the persistence of bits.

Physical destruction always works as a method of data deletion. One of us uses a hammer; another of us prefers his axe. Alas, these methods, while effective, do not meet contemporary standards for recovery and recycling of potentially toxic materials.

Can Data Be Deleted Permanently?

COPIES MAKE DATA HARD TO DELETE

If your computer has ever been connected to a network, destroying its data will not get rid of copies of the same information that may exist on other machines. Your emails went to and from other people—who may have copies on their machines, and may have shared them with others. If you use Google's Gmail, Google may have copies of your emails even after you have deleted them. If you ordered some merchandise online, destroying the copy of the invoice on your personal computer certainly won't affect the store's records.

Rumors arise every now and then that engineers equipped with very sensitive devices can tell the difference between a 0 that was written over a 0 on a disk and a 0 that was written over a 1. The theory goes that successive writing operations are not perfectly aligned in physical space—a “bit” has width. When a bit is rewritten, its physical edges may slightly overlap or fall short of its previous position, potentially revealing the previous value. If such microscopic misalignments could be detected, it would be possible to see, even on a disk that has been zeroed, what the bits were *before* it was zeroed.

No credible authentication of such an achievement has ever been published, however, and as the density of hard disks continues to rise, the like-

lihood wanes that such data recovery can be accomplished. On the other hand, the places most likely to be able to achieve this feat are government intelligence agencies, which do not boast of their successes! So all that can be said for certain is that recovering overwritten data is within the capabilities of at most a handful of organizations—and if possible at all, is so difficult and costly that the data would have to be extraordinarily valuable to make the recovery attempt worthwhile.

How Long Will Data Really Last?

As persistent as digital information seems to be, and as likely to disclose secrets unexpectedly, it also suffers from exactly the opposite problem. Sometimes electronic records become unavailable quite quickly, in spite of best efforts to save them permanently.

Figure 3.17 shows an early geopolitical and demographic database—the Domesday Book, an inventory of English lands compiled in 1086 by Norman monks at the behest of William the Conqueror. The Domesday Book is one of Britain's national treasures and rests in its archives, as readable today as it was in the eleventh century.



British National Archives.

FIGURE 3.17 The Domesday Book of 1086.

In honor of the 900th anniversary of the Domesday Book, the BBC issued a modern version, including photographs, text, and maps documenting how Britain looked in 1986. Instead of using vellum, or even paper, the material was assembled in digital formats and issued on 12-inch diameter video disks, which could be read only by specially equipped computers (see Figure 3.18). The project was meant to preserve forever a detailed snapshot of late twentieth-century Britain, and to make it available immediately to schools and libraries everywhere.

By 2001, the modern Domesday Book was unreadable. The computers and disk readers it required were obsolete and no longer manufactured. In 15 years, the memory even of how the information was formatted on the disks had been forgotten. Mocking the project's grand ambitions, a British newspaper exclaimed, “Digital Domesday Book lasts 15 years not 1000.”



"Domesday Redux," from Ariadne, Issue 56.

FIGURE 3.18 A personal computer of the mid-1980s configured to read the 12-inch videodisks on which the modern "Domesday Book" was published.

Paper and papyrus thousands of years older even than the original Domesday Book are readable today. Electronic records become obsolete in a matter of years. Will the vast amounts of information now available because of the advances in storage and communication technology actually be usable a hundred or a thousand years in the future, or will the shift from paper to digital media mean the loss of history?

The particular story of the modern Domesday Book has a happy ending. The data was recovered, though just barely, thanks to a concerted effort by many technicians. Reconstructing the data formats required detective work on masses of computer codes (see Figure 3.19) and recourse to data structure books of the period—so that programmers in 2001 could imagine how others would have attacked the same data representation problems only 15 years earlier! In the world of computer science, “state of the art” expertise dies very quickly.

The recovered modern Domesday Book is accessible to anyone via the Internet. Even the data files of the original Domesday Book have been transferred to a web site that is accessible via the Internet.

Dec	Cr	Date	Length	VIDEO MAP Frame NUMBER	GRID REF	Zoom ratio Number	Play back Number	Watches	Search	Zoom ratio Number	Watches	Search
<i>CSMAD2001</i>												
Sept 6	072	4001	6749	0000	0000	6749	0003	0049	0700	6849	0001	0000
9/16	330	4001	6749	0000	0000	6749	0003	0033	0700	6849	0001	0000
1622	668	4001	6749			6749	0003					
2448	668			6749		6749	0003					
(Frames)	0000			6749		6749	0003					
3164	700											
4000	073			6749		6749	0003					
4796	120	2002	1049	0003	0000	6749	0103	7558	0710	7239	0001	
5712	165	1103	7749	0001	0411	6749	0103	7558	0812	0344	0711	
6528	199	H100	1749	0001	0025	6749	0103	7558	0509	3744	0001	
7244	150	2002	7649	0001	0008	6749	0103	7558	0008	0344	0711	
9160	180	1700	7149	0008	2012	6749	0103	8688	0102	0344	0008	
9776	230	2307	7249	0003	0004	6749	0103	8688	0203	CSMAD	0001	
9792	426	0001	0143	2002	0000	6749	0203	8330	0000	PEWA	0000	
10600	530	0001	0143	2000	0000	6749	0203	8330	0000	PEWA	0000	
11516	5001	0001	0143	0001	0119	6749	0203	7244	0000	PEWA	0000	
18732	5001	0001	0143	0001	0000	6749	0203	7244	0000	PEWA	0000	
287048	6700	7749	0007	0000	7249	0301	8688	0005	8688	0001		
31476	6700	C155	3618	0005	0000	6749	0301	9681	0005	0000		
289364	6700	7749	0007	0000	7249	0303	0033	0033	0033	0001		
400912	10178	6700	1618	0011	0000	6749	0303	9680	0000	7558	0001	
N300254	10178											
435744	3220	2002	2020	2020	2020	2020	2020	2020	2020	2020	2020	
436224	3220	2020	2020	at end of section boundary								

FIGURE 3.19 Efforts to reconstruct, shortly after the year 2000, the forgotten data formats for the modern “Domesday Book,” designed less than 20 years earlier.

But there is a large moral for any office or library worker. We cannot assume that the back-ups and saved disks we create today will be useful even ten years from now for retrieving the vast quantities of information they contain. It is an open question whether digital archives—much less the box of disk drives under your bed in place of your grandmother’s box of photographs—will be as permanent as the original Domesday Book. An extraordinary effort is underway to archive the entire World Wide Web, taking snapshots of every publicly accessible web page at period intervals. Can the effort succeed, and can the disks on which the archive is held

PRESERVING THE WEB

The Internet Archive

(www.archive.org) periodically records “snapshots” of publicly accessible web pages and stores them away. Anyone can retrieve a page from the past, even if it no longer exists or has been altered. By installing a “Wayback” button (available from the Internet Archive) on your web browser, you can instantly see how any web page looked in the past—just go to the web page and click the Wayback button; you get a list of the archived copies of the page, and you can click on any of them to view it.

themselves be updated periodically so that the information will be with us forever?

Or would we be wisest to do the apparently Luddite thing: to print everything worth preserving for the long run—electronic journals, for example—so that documents will be preserved in the only form we are *certain* will remain readable for thousands of years?



The digital revolution put the power to document ideas into the hands of ordinary people. The technology shift eliminated many of the intermediaries once needed to produce office memoranda and books. Power over the thoughts in those documents shifted as well. The authority that once accompanied the physical control of written and printed works has passed into the hands of the individuals who write them. The production of information has been democratized—although not always with happy results, as the mishaps discussed in this chapter tellingly illustrate.

We now turn to the other half of the story: how we get the information that others have produced. When power over documents was more centralized, the authorities were those who could print books, those who had the keys to the file cabinets, and those with the most complete collections of documents and publications. Document collections were used both as information choke points and as instruments of public enlightenment. Libraries, for example, have been monuments to imperial power. University libraries have long been the central institutions of advanced learning, and local public libraries have been key democratizing forces in literate nations.

If everything is just bits and everyone can have as many bits as they want, the problem may not be having the information, but finding it. Having a fact on the disk in your computer, sitting a few inches from your eyes and brain, is irrelevant, if what you want to know is irretrievably mixed with billions of billions of other bits. Having the haystack does you no good if you can't find your precious needle within it. In the next chapter, we ask: Where does the power now go, in the new world where access to information means finding it, as well as having it?

Functions

Computer Science Principles

What is a Function?

- In BYOB/SNAP, as well as other programming languages, we can take code blocks and put them in a new block that we can define.
- These new blocks are called **functions** or **procedures** because they perform a function for us, such as the ones that you have already been using in BYOB/SNAP.

What is a Function?

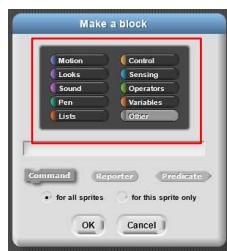
- We can create our script (or code block) once ~ and use it as many times as we want without having to re-write the code.
- We only need to “call” or “invoke” it.
- This is called *Procedural Abstraction*.

Functions in BYOB/SNAP

- To create a function in BYOB/SNAP, we are going to make our own block.
 - Click on **make a block** at the bottom of the **Variables** tab.
- Make a block**
- This will open up the **make a block** dialog box.

Make a Block in BYOB/SNAP

- You get to choose which palette the block will be in.
 - Choose the one that “makes sense”
 - Example
 - Our new block will be to draw a square, so we will choose Motion.



Types of Function Blocks

- There are three (3) types of function blocks you can create in BYOB/SNAP.
 1. **Command**
 - Creates a block that “makes something happen”
 - The same blocks that look like puzzle pieces.
 2. **Reporter**
 - Creates a block that reports or returns a value.
 - These blocks look like some of your operator blocks, such as the () + () block. (rounded ends)
 3. **Predicate**
 - Creates a block that reports or returns either true or false.
 - These blocks look like some of your operator blocks, such as the () < () block. (pointed ends)

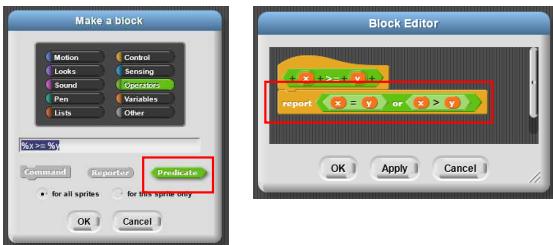
Command Function Blocks

- A command function block is created to have a task carried out without any value being returned.
 - Such as drawing a square.



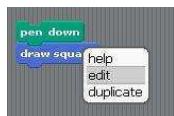
Predicate Function Blocks

- A predicate block returns either true or false.
- You will need to report other blocks that evaluate to either true or false.
- The Block Editor will open with a report block.



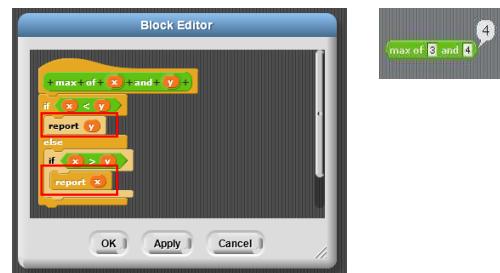
Arguments

- If you are modifying a block to add arguments, right-click on the new block and select **edit** to go back to the block editor.



Reporter Function Blocks

- Reporter blocks can report a value.
- You will need to use the report () block to return or report the value.

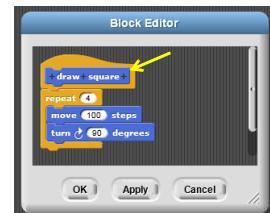


Arguments

- You have created a block that draws a square, but it only draws a square where each side is of length 100 steps.
- It would be great if we could specify how long we wanted each side to be.
- We will edit the block to accept an **argument** (or **input**), which tells it the length of the square it has to draw.

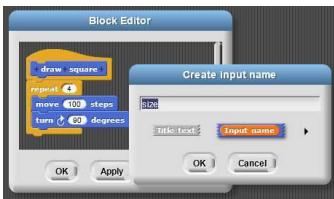
Arguments

- In the Block Editor, notice that when you move the mouse over the top row of the new block, some plus signs (+) show up.
- When you click on these plus signs, you can add more text or arguments.
- When you click on the text between the plus signs, you can delete or modify that text.
- Click on the plus sign at the far right as shown below:



Arguments

- When you click on the plus sign on the far right, you should get the following dialog box.
- With this dialog box, we can select if we want to add input (orange) or more text (blue).
- We want to add the input size, so we type size, select Input Name and click OK.



Arguments

- Now, we have a variable inside our block definition.
- Drag the variable size down into the move block.
- Whenever we need a new copy of a variable, we just grab the copy from that variable in the top row.



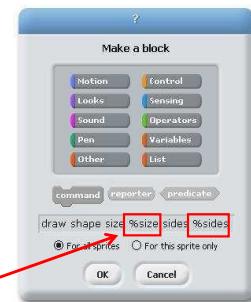
Arguments

- When we click OK, we will see that our draw square block now takes an argument.
- We can put different numbers in the blank and draw squares of different sizes!



Arguments

- If you are just starting to create your function block, you can create the inputs to this block in exactly the same way as we did in the previous section, by clicking on the plus signs to add input.
- You can also type the names of the input as shown.
 - The percent signs (%) indicate that the word should be an input.



Script Variables

- A script variable is a variable that is only available inside that specific script.
- After you change the name, simply drag it into the Block Editor.
- You can then click and drag the variable's name from the script variables block into spots, such as the report box.



Input Types

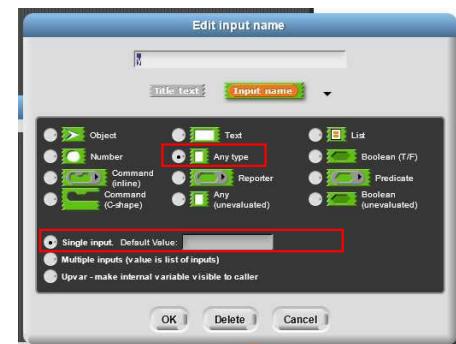
- You can limit the types of values that can be entered into a block's argument.
- Open the Block Editor for the function block and click on the input's name.
- The following window displays.



Input Types

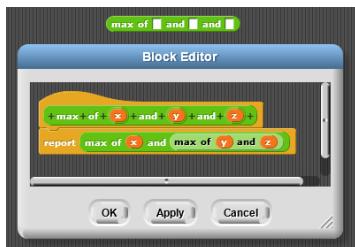
- Then, click on the right arrow in the pop-up box.
- This will open the dialog box shown on the next slide.
- This allows us to specify the shape of the slot.
- We want a numbers-only slot (as shown selected below).
- We can also specify that we want the variable to have a *default* value.
- This is similar to blocks like move that always start out with the default value 10.

Input Types



Composition of Functions

- Our custom-made blocks are blocks like any other, and we can use them in other block definitions.



Different Kinds of Variables

- Normal/Global Variables
- Sprite Specific Variables
- Arguments to a function
- Block Variables
- Script Variables

Normal/Global Variables

- These variables are made in the regular menu and can be used ANYWHERE!
- The variable "score" is an example.
- These can be used by any sprite, in any block or in any script.



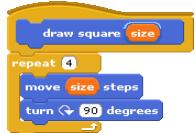
Sprite Specific Variables

- When you create a "normal/global" variable you can select that the variable is "For this sprite only".
- Then these variables will show up as variables listed below the line in the variables tab.
- We recommend *not* using these variables in blocks.



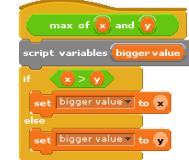
Arguments to a function

- A variable set by the person calling the function.
- We also refer to this as "**input**".
- *This can ONLY be used within the block editor.*



Script Variables

- The "script variable" block gives us a variable that we can use inside of this script.
- These *can only be used in that particular script*.
- The script could be a block script (shown below) or a regular script.



Curriculum (/bjc-course/curriculum) / Unit 3 (/bjc-course/curriculum/03-build-your-own-blocks) /

Lab 1 (/bjc-course/curriculum/03-build-your-own-blocks/labs/01-build-your-own-blocks) /

Lab: Build Your Own Blocks

Let's open up SNAP at <http://snap.berkeley.edu/run> (<http://snap.berkeley.edu/run>)

As you have seen, SNAP/BYOB is a powerful language that has a substantial repository of useful blocks for a variety of purposes. However, as you may have noticed, SNAP/BYOB does not have all the blocks that you may need, and often, it would prove useful if we could create new blocks.

Make Your Own Block: A Tutorial

We are going to teach the computer how to draw a square using a block named `draw square`. Please follow the steps below:

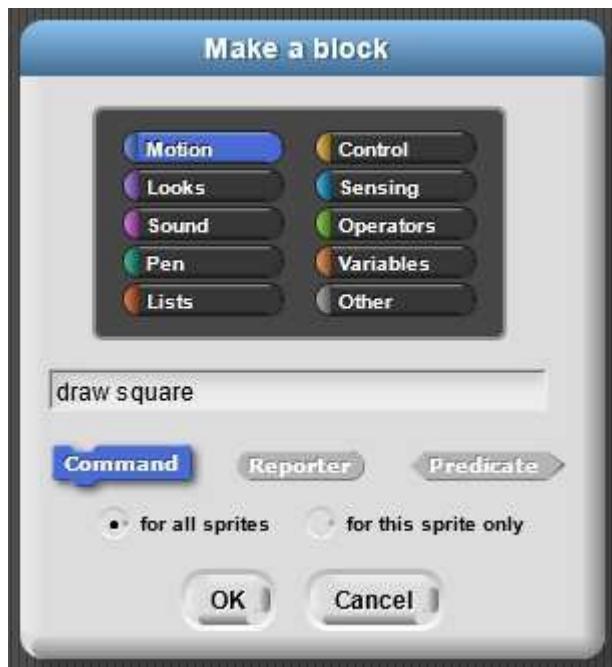
- Click on make a block at the bottom of the variables tab.



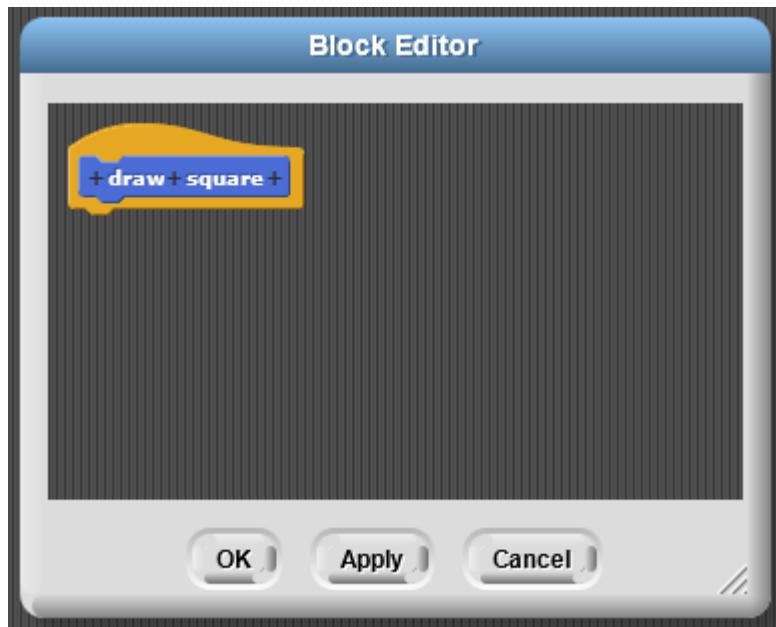
- This will open up the `make a block` dialog box. Now, you get to choose which tab the block should go into. Our block is going to draw a square, so let us choose `Motion`.



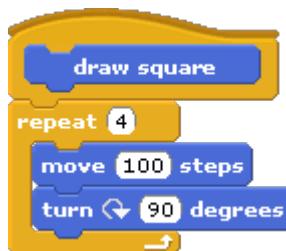
- When we selected `Motion`, the block became blue. We now have the option of making blocks of different shapes. Right now, however, we are just going to make a (regular) command block.



- When we click **OK**, we should see the block editor below.



- Use the blocks from the regular menus to create a script that draws a square, as shown below.



- When you click **OK**, you should be able to use this block as if it were a regular block. Since you created the block as a **Motion** block, it will end up at the bottom of the **Motion** tab.



Congratulations! You have just created your own block!

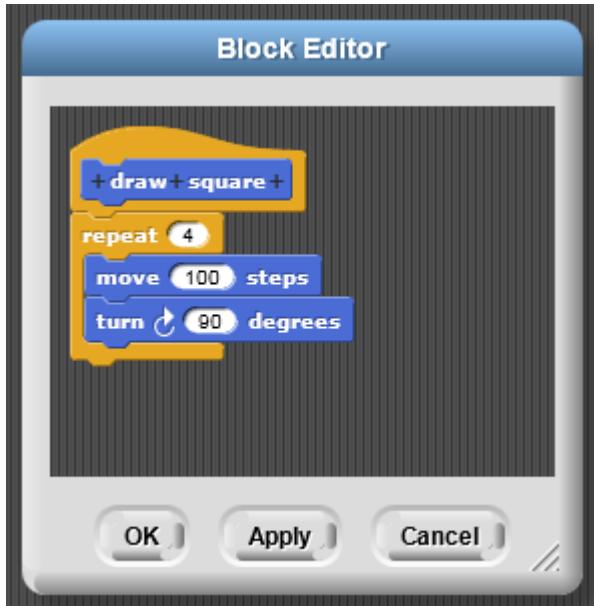
Improving the draw square block

You have created a block that draws a square, but it only draws a square where each side is of length `100` steps. It would be great if we could specify how long we wanted each side to be. We will edit the block to accept an *argument* (or *input*), which tells it the length of the square it has to draw.

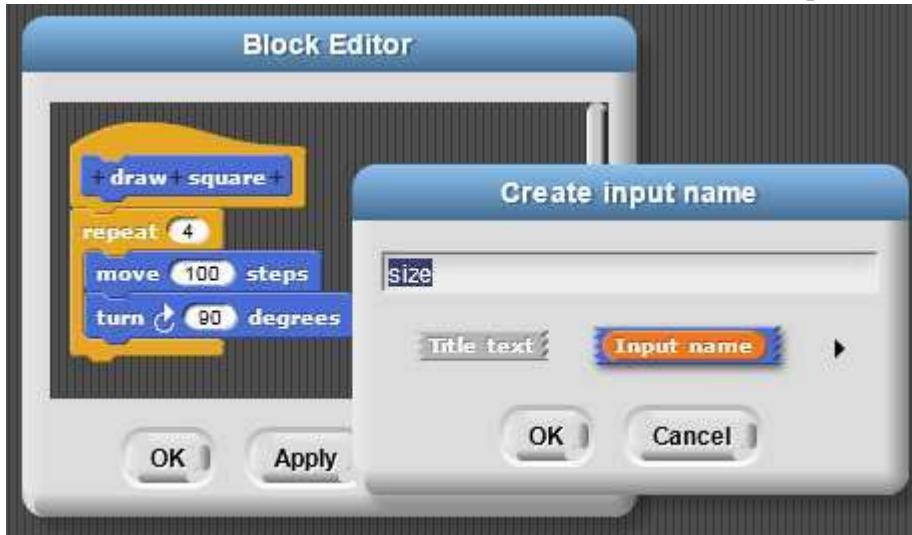
- We are going to go back and edit the block. Right-click on the new block and select edit to go back to the block editor.



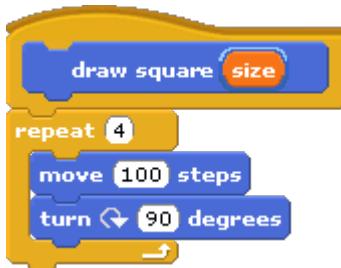
- In the `Block Editor`, notice that when you move the mouse over the top row of the new block, some plus signs (+) show up. When you click on these plus signs, you can add more text or arguments. When you click on the text between the plus signs, you can delete or modify that text. Click on the plus sign at the far right as shown below:



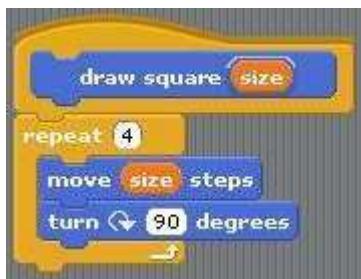
- When you click on the plus sign on the far right, you should get the following dialog box. With this dialog box, we can select if we want to add input (orange) or more text (blue). We want to add the input `size`, so we type `size`, select `Input Name` and click `OK`.



- Now, we have a variable inside our block definition.



- Drag the variable `size` down into the move block. Whenever we need a new copy of a variable, we just grab the copy from that variable in the top row.



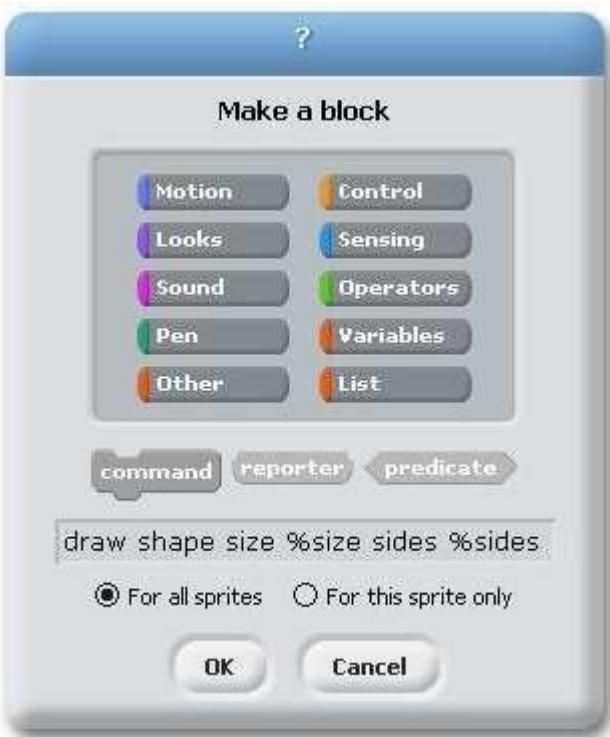
- When we click OK, we will see that our draw square block now takes an argument. We can put different numbers in the blank and draw squares of different sizes!



Make a draw shape Block

Now, you are going to make a block that takes two inputs. We want to create a `draw shape` block that takes a number of sides and a number of pixels for the length of each side. We will call these input arguments `n` and `pixel`. This exercise should be done with a partner, so introduce yourself to your neighbor and get started on the exercise together!

By the way, you can create the inputs to this block in exactly the same way as we did in the previous section, by clicking on the plus signs to add input; however, you can also type the names of the input as shown below.



The percent signs (%) indicate that the word should be an input. We want you to feel comfortable with both entry methods.

The Max block

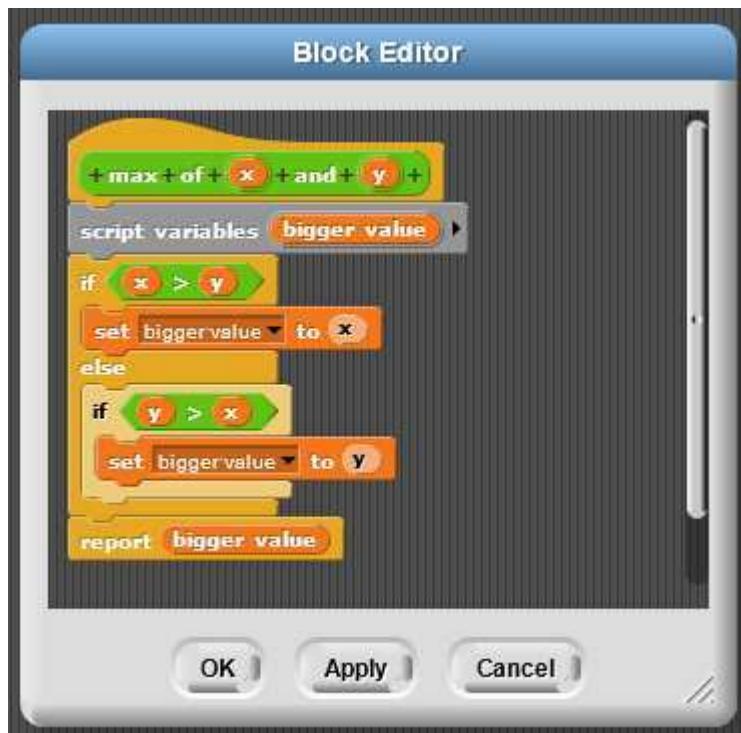
We will now make a different kind of block – a *reporter* block. To demonstrate this, we will make a block called `max` that takes two numbers as input and reports the bigger value (the maximum).

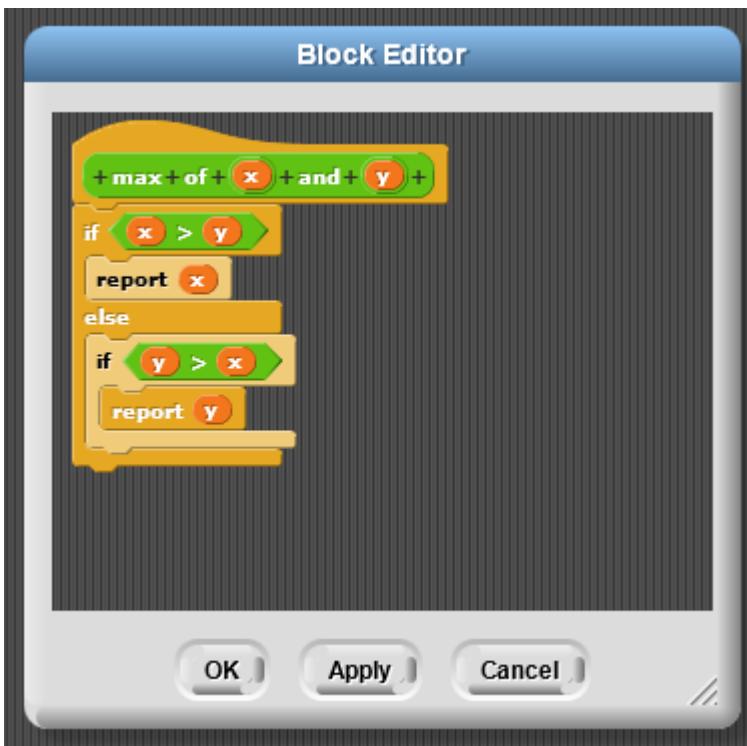


- Click `Make a block` and select the `Operators` tab. We want a reporter block. This will give the block its round shape as shown above. As the name implies, reporter blocks can report a value. In the image below, you can see that we used the % shortcut for making input variables.



- This should give you a blank Block editor. We need to figure out what should be reported. To keep track of the value to be reported, we are going to make another variable. There are two ways to do this: Use a `Script Variable` block. You can click on the name of the variable and change it to bigger value. Alternatively, you can just report which of the two is larger.





Input Types

Unfortunately, we have a bug with our `max` block! We wanted the `max` block to work only for numbers. Yet, you can type text in!



We are going to limit the `max` block to accept only numbers as arguments.

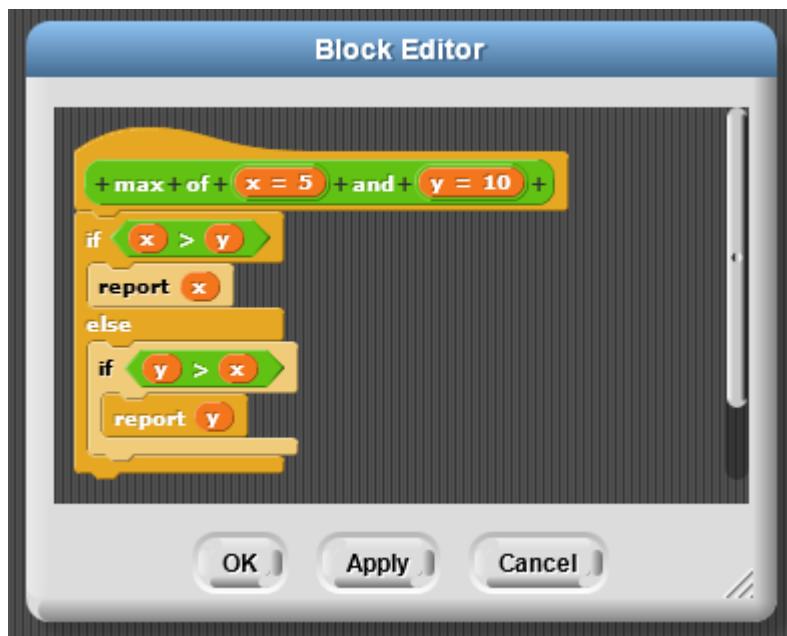
- Open the Block Editor for the `max` block and click on the input `x`



- Then, click on the right arrow in the pop-up box shown above. This will open the dialog box shown below. This allows us to specify the shape of the slot. We want a numbers-only slot (as shown selected below). We can also specify that we want the variable to have a *default* value; this is similar to blocks like move that always start out with the default value 10.



- Modify both the `x` and `y` variables to take only numbers and to have the default values of `5` and `10`, respectively. Your `Block Editor` should then look like this, with the default values shown in the header.



- When you click OK, you should be able to see your block in the Operators tab, with the default values filled in. Also, note that you will no longer be able to enter text.



Note: Maybe we *did* want the `max` block to work with words! However, for the `draw square` and `draw shape` blocks, we definitely only wanted numbers. Modify those blocks to only take in numbers.

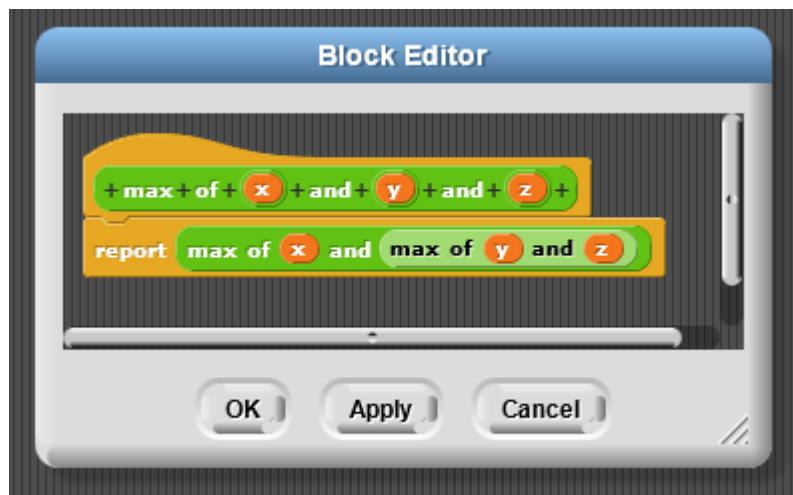
Composition of Functions

Our custom-made blocks are blocks like any other, and we can use them in other block definitions. To demonstrate

this, we are going to make a block that computes the maximum of three values



Repeat the steps from the last tutorial to make this version of the `max` block also only take numbers. Then, in the script for the block, we can use two copies of our `max` block.



Try It! AddNumbers and JoinText

Your challenge is to create the following two blocks. You can use the same project file.

- A three-argument addition operator that only accepts numbers.



- A three-argument `join` operator that has the default values shown below and only accepts text.



Predicates

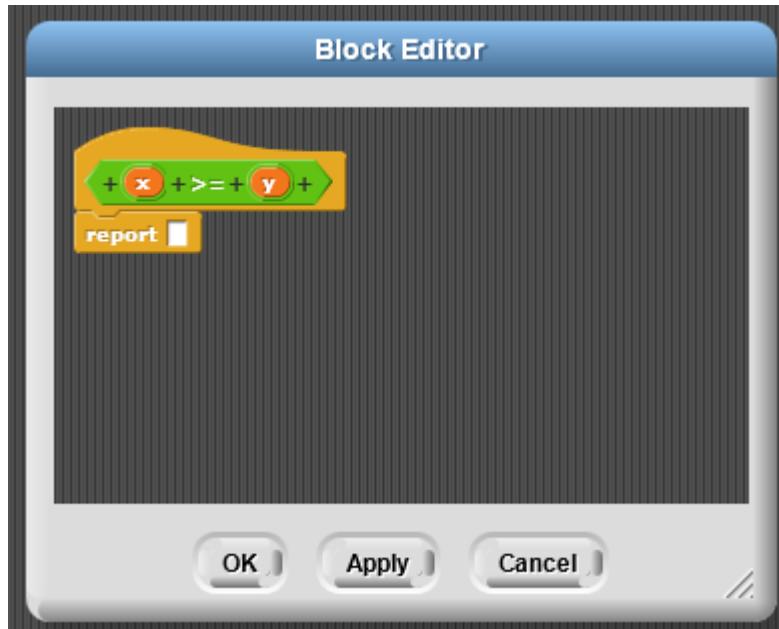
We want to make our own predicate, a kind of block that reports either `true` or `false`. We have a “greater than” operator (`>`), an “equal” operator (`=`), and a “less than” operator (`<`), but we want a new “greater than or equal to” (`=`)

operator.

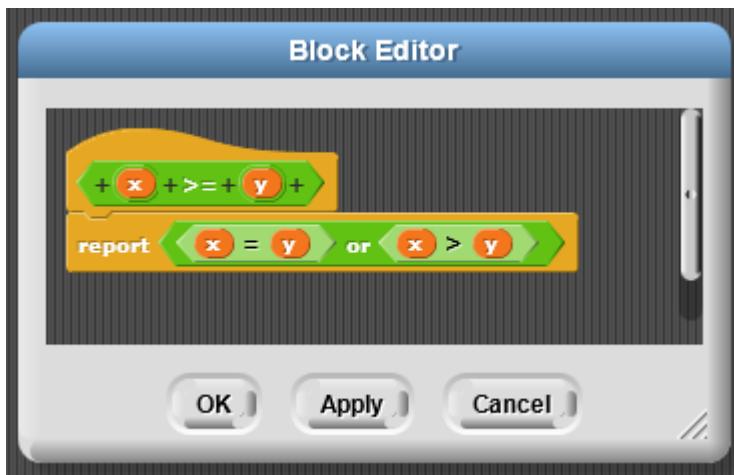
- We will create the new block and select the **predicate** shape.



- This gives a **Block Editor** that has a predicate-shaped blank at the bottom. There, we need to place the predicate that we want to report.



- We can fill that in with a composition of a “greater than”, “or”, and “equal” operators. Make this with your partner and then try it out. Notice that this predicate block reports either **true** or **false**.



Try It! Predicates: Make a between block

Create a new predicate block that determines if a number is between two other numbers. The block should return `true` if the first number is between the two numbers or if it is equal to either of the numbers.

`is [] between [] and []`

Different Kinds of Variables

We've seen a lot of different types of variables.

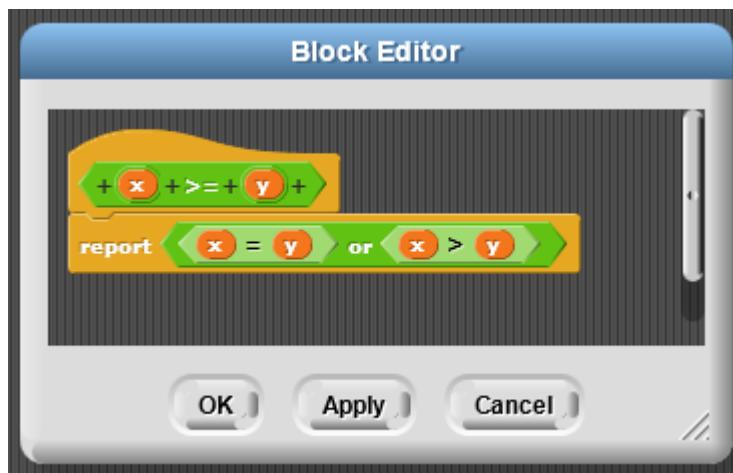
- **Normal/Global variables:** These variables are made in the regular menu and can be used ANYWHERE! The variable "score" below is an example. *These can be used by any sprite, in any block or in any script.*



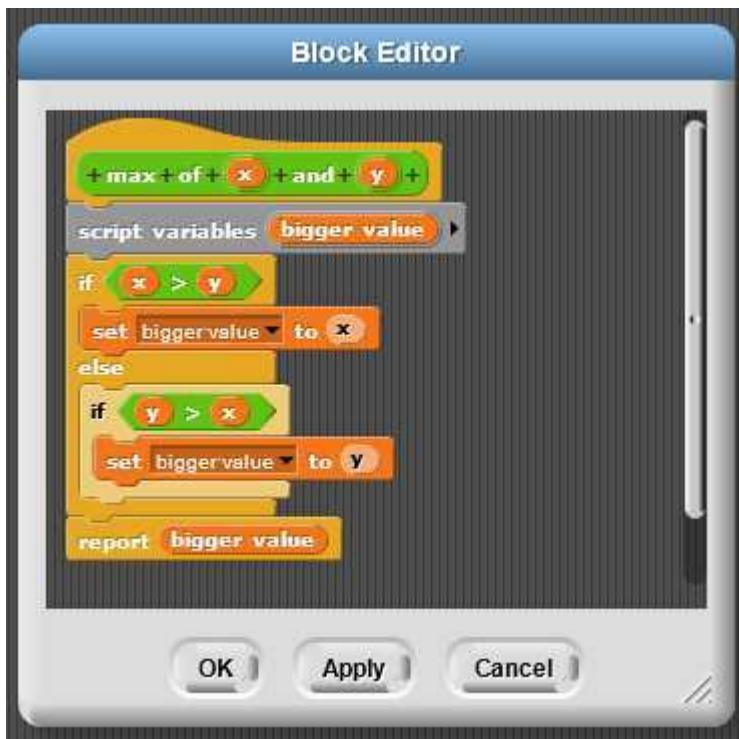
- **Sprite Specific Variables:** When you create a "normal/global" variable you can select that the variable is "For this sprite only". Then these variables will show up as variables listed below the line in the variables tab. *We recommend not using these variables in blocks.*



- **Arguments to a function:** A variable set by the person calling the function. We also refer to this as “input”. *This can ONLY be used within the block editor.*



- **Script Variables:** The “script variable” block gives us a variable that we can use inside of this script. *These can only be used in that particular script. The script could be a block script (shown below) or a regular script.*



Try It!: Simplifying a tic-tac-toe board drawer using functions

In the previous lab, we used a `repeat` block to avoid duplicating code. Similarly, we can use a function to avoid duplicating code. Below is some code to draw a tic-tac-toe board. Your goal is to create functions that make this code simpler. One important thing to keep in mind is to give your new functions really intuitive names, so that it is easy to read the code and understand what it does.

Using the blocks below, create two new function blocks, `draw line` and `next line`, to draw the tic-tac-toe board. You will still have some of the code from the original script. You can create additional blocks for those functions.

The Scratch script consists of a repeating loop with a control `repeat () []`. Inside the loop, the script performs the following steps:

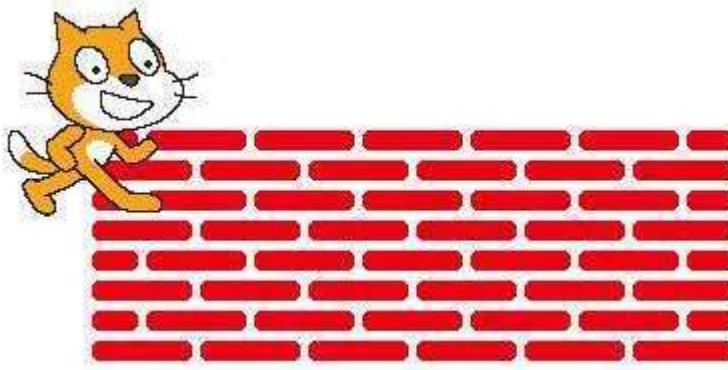
- Set pen color to blue
- Set pen size to 1
- Set background color to black
- Set stage background to "solid black" and stage size to 400x400
- Set stage speed to 10
- Set x position to -200 and y position to 50
- Point in direction 90°
- Clear the stage
- Pen down
- Move 300 steps
- Move -300 steps
- Pen up
- Turn 90 degrees counter-clockwise
- Move 100 steps
- Turn 90 degrees counter-clockwise
- Pen down
- Move 300 steps
- Move -300 steps
- Pen up
- Move 100 steps
- Turn 180 degrees counter-clockwise
- Pen down
- Move 300 steps
- Move -300 steps
- Pen up
- Turn 90 degrees counter-clockwise
- Move 100 steps
- Turn 90 degrees counter-clockwise
- Pen down
- Move 300 steps
- Move -300 steps
- Pen up

Curriculum (/bjc-course/curriculum) / Unit 3 (/bjc-course/curriculum/03-build-your-own-blocks) / Lab 2 (/bjc-course/curriculum/03-build-your-own-blocks/labs/02-brick-wall-project) /

Project: Brick Wall

Let's open up SNAP at <http://snap.berkeley.edu/run> (<http://snap.berkeley.edu/run>)

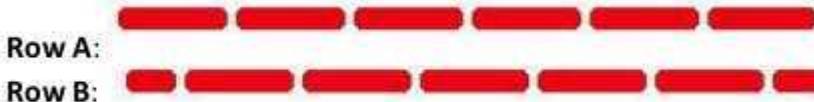
Sometimes, when we write programs and scripts, it feels like we have hit a brick wall! (This is a good sign - it is supposed to be hard!) We are going to draw this brick wall.



This project is not just about drawing; it is also about practicing abstraction. You will draw the following brick wall by implementing the blocks listed below.

Note: You must implement these blocks and adhere to the abstraction described below.

There are two kinds of rows in this brick wall:



The big idea is that there are three levels of abstraction. Start creating your new blocks at Level 3, then create for Level 2 (using your new blocks from Level 1), and then create Level 1 using your Level 2 blocks.

Note: a “brick” is just a thick line.

At the lowest level of abstraction (Level 3):

- You need to figure out how to draw individual bricks, small bricks and spaces. The bricks are simply thick lines.
- This level of abstraction contains the following blocks:
 - The `Draw Brick` block, which draws a single brick.
 - The `Draw Small Brick` block, which draws the small brick for the edges of row B. Note that this brick will not be exactly half as long as the full brick. Part of this assignment is figuring out how long the “small brick” should be.
 - The `Draw Space` block, which draws a space between each brick or small brick.

At the middle level of abstraction (Level 2):

- You can use the functionality provided by the bottom level of abstraction to make entire rows of bricks.
- The rows referred to as “Row A” and “Row B” should look like the rows shown above.

- This level of abstraction contains the following blocks:
- The `Initialize Pen` block, which should initialize the pen color and size.
- The `Initialize Character Position` and `Direction` block, which should initialize the position and direction of the character.
- The `Draw Row A` block, which should draw a single copy of Row A.
- The `Draw Row B` block, which should draw a single copy of Row B.
- The `Transition between Row A and B with __ space` block, which should transition between the end of Row A and the beginning of Row B, leaving a space as wide as the number of pixels specified by the input argument.
- The `Transition between Row B and A with __ space` block, which should transition between the end of Row B and the beginning of Row A, leaving a space as wide as the number of pixels specified by the input argument.

HINT: You will need to determine if there is an even number of rows or an odd number of rows to be drawn as well as if the row being drawn is an even number or odd number one. You can use the `() mod ()` block to help.

Example: if `TotalNumberOfRows mod 2 = 0` then the total number of rows is even (any even number divided by 2 leaves a remainder of 0). You can use the same logic to determine if the row to be drawn is even or odd (`if RowNumber mod 2 = 0`).

At the highest level of abstraction (level 1)

- You will put together the full brick wall using only the functionality provided by the middle level of abstraction.
- This level of abstraction contains only the `Draw a Brick Wall with __ rows` block, which draws a brick wall with the specified number of rows.

Note: Whenever you need to refer to a number in the program, use a variable. This is generally considered good style, because you can use the same variable in multiple places in your program, and you only need to change the value of the variable to change it in multiple places at once.

In summary, you should implement the following blocks:

Draw a Brick Wall with rows▼
Level 1

||

Initialize pen

▼

Level 2

||

Initialize character position and direction**Draw Row A****Draw Row B****Transition between Row A and B with spaces****Transition between Row B and A with spaces****Draw Brick**

▼

Level 3

||

Draw Half Brick**Draw Space**

Unit 4: Conditionals and Artificial Intelligence

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts
- 3: The student can use computing tools and techniques for creative expression.
- 4: The student can use programming as a creative tool.
- 16: The student can express an algorithm in a language.
- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 19: The student can explain how programs implement algorithms.
- 22: The student can develop a correct program.
- 23: The student can employ appropriate mathematical and logical concepts in programming.
- 30: The student can analyze the beneficial and harmful effects of computing.
- 31: The student can connect computing within economic, social, and cultural contexts.

Readings/Lectures

- Lecture Slides 4.01: Functions (/bjc-course/curriculum/04-conditionals-and-ai/readings/01-conditional-blocks-slides.pdf)
- Reading 4.02: How many days are in a month? (Example) (/bjc-course/curriculum/04-conditionals-and-ai/readings/02-days-in-month-example)
- Lecture Slides 4.03: AI (/bjc-course/curriculum/04-conditionals-and-ai/readings/03-ai-slides.pdf)
- Lecture Video 4.04: If Blocks (/bjc-course/curriculum/04-conditionals-and-ai/readings/04-if-blocks-video.mp4)

External Links

- Cleverbot (<http://www.cleverbot.com/>)
- What is Watson? (http://www.nytimes.com/2010/06/20/magazine/20Computer-t.html?_r=3&) - *New York Times*
- Watson on Jeopardy (<https://www.youtube.com/watch?v=seNkjYyG3gI>) - *Youtube*
- Computers Solve Checkers (<https://www.scientificamerican.com/article.cfm?id=computers-solve-checkers-its-a-draw>) - *Scientific American*
- Who is ASIMO? (<http://asimo.honda.com/asimotv/>)
- The Google Car (<http://www.forbes.com/sites/chunkamui/2013/01/22/fasten-your-seatbelts-googles-driverless-car-is-worth-trillions/>) - *Forbes*

Labs/Exercises

- Lab 4.01: Random Numbers and Conditionals: Guessing Game (/bjc-course/curriculum/04-conditionals-and-ai/labs/01-random-numbers-and-conditionals)

Conditional Statements

CS Principles

THE IF AND IF...ELSE STRUCTURES

If and If...Else Structures

- Also known as a **Conditional Statement**
- Allows a sprite to make a decision
- Can use sensing tiles to test conditions
- Predicate fields are used for conditions that produce either true or false



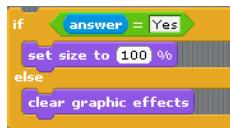
When to use the If Structure

- When the sprite has to make a decision between performing a block of code or not – Yes/No, True/False



When to use the If...Else Structure

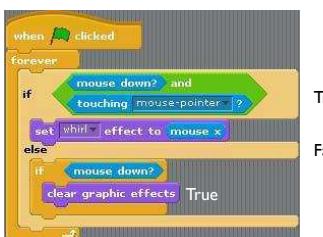
- When the sprite has to make a decision between either performing one block of code or performing a different block of code



NESTED IF'S AND COMPOUND CONDITIONS

Nested If Statements

- When one If-structure is used in another IF-structure, this new structure is known as a **Nested IF**.
- The Nester IF checks two or more conditions.



Logical Operators

- Allow us to put two or more conditions together to create one compound condition
- And, Or and Not
- Order of Precedence: Not, And, Or

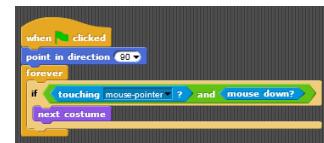


Conditional Operator Example

- All three can be used in any combination.
- Be careful – Don't get confusing. Try to keep them as simple as you can.

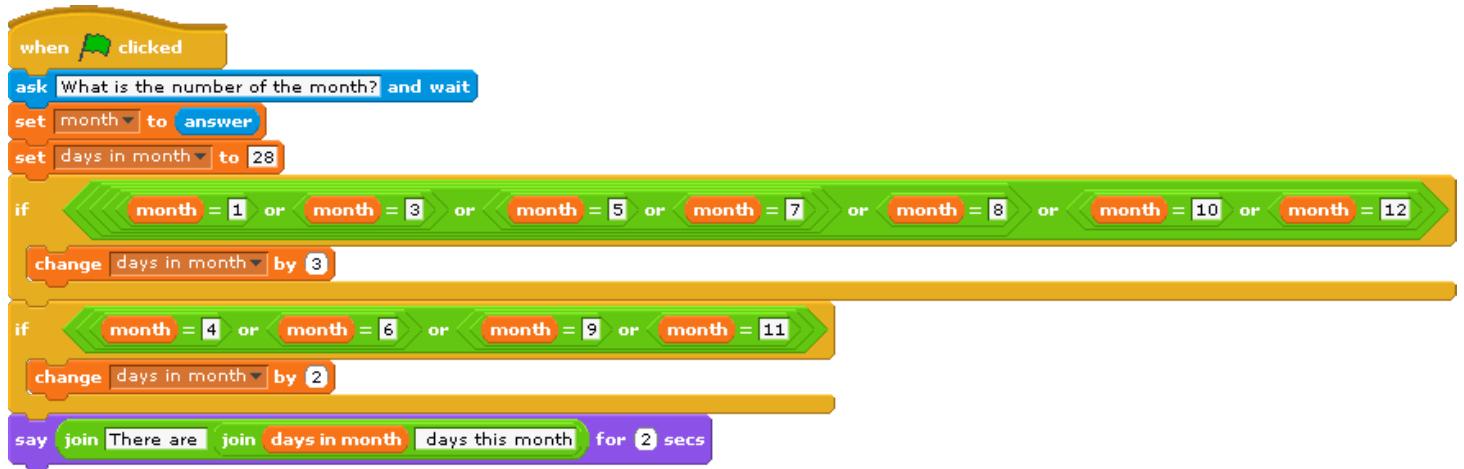


Conditional Examples



How many days are there in a given month?

We have created a Scratch file (..//code/DaysInMonth.sb) that calculates the number of days in a month based on a month number(eg. 1-12) that a user types in. Discuss with your partner how this Scratch file works. There are a lot of new things in this program that you may not have seen before, so be sure to play around with them.



Note: This code is not written with the best style. We hope your reaction to this is YUCK! We have made the code explicitly complex to help you review some of the blocks from the first lab. In future lab sessions, we will learn to write this type of script more elegantly.

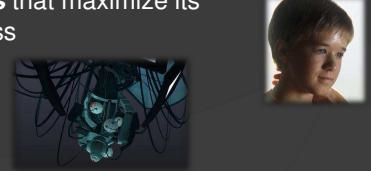
ARTIFICIAL INTELLIGENCE



Artificial Intelligence

- ◎ What is Artificial Intelligence?
 - The Study and Design of Intelligent Agents

- ◎ What is an Intelligent Agent?
 - A system that **perceives** its environment and **takes actions** that maximize its chances of success



Artificial Intelligence

- ◎ What is the **environment** of the Tic Tac Toe paper?

- ◎ What **actions** did it take to maximize its chances of success?

Artificial Intelligence Philosophies

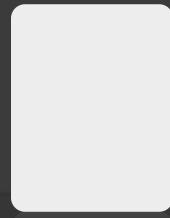
- ◎ Turing
 - If a machine acts as intelligently as a human being, then it is as intelligent as a human being.
 - Judge intelligence of a machine based on its behavior.

Artificial Intelligence Philosophies

- ◎ Searle
 - “The appropriately programmed computer with the right inputs and outputs would thereby have a mind in exactly the same sense human beings have minds.”
 - Strong AI – a correct simulation of a mind, IS a mind
 - Machines can act intelligently, and understand in the same sense people do
 - Weak AI – a correct simulation is a model of the mind
 - Machines can only act intelligently, and thus only mimic human behavior

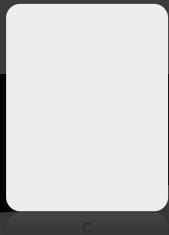
Gender Guessing Game

- ◎ Example) Determine the gender of someone in another room by conversing with them.
 - Can't ask direct answer questions (i.e. are you male/female, etc.)



Turing Test

- Measure of a computers ability to demonstrate intelligence.
- Consider the Gender Game again...



Turing Test

- If the observer cannot reliably tell the machine from a human, the machine is said to be “intelligent”.
- If a machine is intelligent, it can “think”

Turing Test

- Ultimately says...
- Can computers mimic human behavior?
- This is the goal of AI

Chinese Room Test

- You write a statement in Chinese characters on a piece of paper and slide it under the door.
- You receive a response back to your statement in Chinese.
- Continue until bored.



Chinese Room Test

If you can carry on an intelligent conversation using a piece of paper slid under a door, does this imply that something inside the room understands what you are saying?

Chinese Room Test

- Who understands Chinese?
- What is the input in this test?
- What is the environment?

Strong AI vs. Weak AI

- ◎ Strong AI

- The machine ***actually*** understands Chinese

- ◎ Weak AI

- The machine ***simulates*** understanding Chinese

Examples

- ◎ Plen

- http://www.youtube.com/watch?v=C_UVklCbThI
- <http://www.youtube.com/watch?v=quzwHqjgrDw&NR=1>
- <http://www.youtube.com/watch?v=dnBbev4-VcE&feature=related>

Examples

- ◎ Asimo

- <http://www.youtube.com/watch?v=NZnqYDDDFW4>
- http://www.youtube.com/watch?v=_PMw023nS48&feature=related
- <http://www.youtube.com/watch?v=0tRo6a4VhvU>

Examples

- ◎ Google Cars

- <http://www.youtube.com/watch?v=X0l5DHOETFE>

Non-Robotics Examples

- ◎ Checkers

- ◎ Deep Blue

- ◎ Watson

- <http://www.youtube.com/watch?v=ZLdkJpAtt1>
- 2:00

This copy is for your personal, noncommercial use only. You can order presentation-ready copies for distribution to your colleagues, clients or customers [here](#) or use the "Reprints" tool that appears next to any article. Visit [www.nytreprints.com](#) for samples and additional information. [Order a reprint of this article now.](#)

June 16, 2010

What Is I.B.M.'s Watson?

By CLIVE THOMPSON

"Toured the Burj in this U.A.E. city. They say it's the tallest tower in the world; looked over the ledge and lost my lunch."

This is the quintessential sort of clue you hear on the TV game show "Jeopardy!" It's witty (the clue's category is "Postcards From the Edge"), demands a large store of trivia and requires contestants to make confident, split-second decisions. This particular clue appeared in a mock version of the game in December, held in Hawthorne, N.Y. at one of I.B.M.'s research labs. Two contestants — Dorothy Gilmartin, a health teacher with her hair tied back in a ponytail, and Alison Kolani, a copy editor — furrowed their brows in concentration. Who would be the first to answer?

Neither, as it turned out. Both were beaten to the buzzer by the third combatant: Watson, a supercomputer.

For the last three years, I.B.M. scientists have been developing what they expect will be the world's most advanced "question answering" machine, able to understand a question posed in everyday human elocution — "natural language," as computer scientists call it — and respond with a precise, factual answer. In other words, it must do more than what search engines like Google and Bing do, which is merely point to a document where you might find the answer. It has to pluck out the correct answer itself. Technologists have long regarded this sort of artificial intelligence as a holy grail, because it would allow machines to converse more naturally with people, letting us ask questions instead of typing keywords. Software firms and university scientists have produced question-answering systems for years, but these have mostly been limited to simply phrased questions. Nobody ever tackled "Jeopardy!" because experts assumed that even for the latest artificial intelligence, the game was simply too hard: the clues are too puzzling and allusive, and the breadth of trivia is too wide.

With Watson, I.B.M. claims it has cracked the problem — and aims to prove as much on national TV. The producers of "Jeopardy!" have agreed to pit Watson against some of the game's best former players as early as this fall. To test Watson's capabilities against actual humans, I.B.M.'s scientists began holding live matches last winter. They mocked up a conference room to resemble the actual "Jeopardy!" set, including buzzers and stations for

the human contestants, brought in former contestants from the show and even hired a host for the occasion: Todd Alan Crain, who plays a newscaster on the satirical Onion News Network.

Technically speaking, Watson wasn't in the room. It was one floor up and consisted of a roomful of servers working at speeds thousands of times faster than most ordinary desktops. Over its three-year life, Watson stored the content of tens of millions of documents, which it now accessed to answer questions about almost anything. (Watson is not connected to the Internet; like all "Jeopardy!" competitors, it knows only what is already in its "brain.") During the sparring matches, Watson received the questions as electronic texts at the same moment they were made visible to the human players; to answer a question, Watson spoke in a machine-synthesized voice through a small black speaker on the game-show set. When it answered the Burj clue — "What is Dubai?" ("Jeopardy!" answers must be phrased as questions) — it sounded like a perkier cousin of the computer in the movie "WarGames" that nearly destroyed the world by trying to start a nuclear war.

This time, though, the computer was doing the right thing. Watson won \$1,000 (in pretend money, anyway), pulled ahead and eventually defeated Gilmartin and Kolani soundly, winning \$18,400 to their \$12,000 each.

"Watson," Crain shouted, "is our new champion!"

It was just the beginning. Over the rest of the day, Watson went on a tear, winning four of six games. It displayed remarkable facility with cultural trivia ("This action flick starring Roy Scheider in a high-tech police helicopter was also briefly a TV series" — "What is 'Blue Thunder'?"), science ("The greyhound originated more than 5,000 years ago in this African country, where it was used to hunt gazelles" — "What is Egypt?") and sophisticated wordplay ("Classic candy bar that's a female Supreme Court justice" — "What is Baby Ruth Ginsburg?").

By the end of the day, the seven human contestants were impressed, and even slightly unnerved, by Watson. Several made references to Skynet, the computer system in the "Terminator" movies that achieves consciousness and decides humanity should be destroyed. "My husband and I talked about what my role in this was," Samantha Boardman, a graduate student, told me jokingly. "Was I the thing that was going to help the A.I. become aware of itself?" She had distinguished herself with her swift responses to the "Rhyme Time" puzzles in one of her games, winning nearly all of them before Watson could figure out the clues, but it didn't help. The computer still beat her three times. In one game, she finished with no money.

"He plays to win," Boardman said, shaking her head. "He's really not messing around!" Like most of the contestants, she had started calling Watson "he."

WE LIVE IN AN AGE of increasingly smart machines. In recent years, engineers have pushed into areas, from voice recognition to robotics to search engines, that once seemed to be the preserve of humans. But I.B.M. has a particular knack for pitting man against machine. In 1997, the company's supercomputer Deep Blue famously beat the grandmaster [Garry Kasparov](#) at chess, a feat that generated enormous publicity for I.B.M. It did not, however, produce a marketable product; the technical accomplishment — playing chess really well — didn't translate to real-world business problems and so produced little direct profit for I.B.M. In the mid '00s, the company's top executives were looking for another high-profile project that would provide a similar flood of global publicity. But this time, they wanted a "grand challenge" (as they call it internally), that would meet a real-world need.

Question-answering seemed to be a good fit. In the last decade, question-answering systems have become increasingly important for firms dealing with mountains of documents. Legal firms, for example, need to quickly sift through case law to find a useful precedent or citation; help-desk workers often have to negotiate enormous databases of product information to find an answer for an agitated customer on the line. In situations like these, speed can often be of the essence; in the case of help desks, labor is billed by the minute, so high-tech firms with slender margins often lose their profits providing telephone support. How could I.B.M. push question-answering technology further?

When one I.B.M. executive suggested taking on "Jeopardy!" he was immediately pooh-poohed. Deep Blue was able to play chess well because the game is perfectly logical, with fairly simple rules; it can be reduced easily to math, which computers handle superbly. But the rules of language are much trickier. At the time, the very best question-answering systems — some created by software firms, some by university researchers — could sort through news articles on their own and answer questions about the content, but they understood only questions stated in very simple language ("What is the capital of Russia?"); in government-run competitions, the top systems answered correctly only about 70 percent of the time, and many were far worse. "Jeopardy!" with its witty, punning questions, seemed beyond their capabilities. What's more, winning on "Jeopardy!" requires finding an answer in a few seconds. The top question-answering machines often spent longer, even entire minutes, doing the same thing.

"The reaction was basically, 'No, it's too hard, forget it, no way can you do it,' " David Ferrucci told me not long ago. Ferrucci, I.B.M.'s senior manager for its Semantic Analysis and Integration department, heads the Watson project, and I met him for the first time last November at I.B.M.'s lab. An artificial-intelligence researcher who has long specialized in

question-answering systems, Ferrucci chafed at the slow progress in the field. A fixture in the office in the evenings and on weekends, he is witty, voluble and intense. While dining out recently, his wife asked the waiter if Ferrucci's meal included any dairy. "Is he lactose intolerant?" the waiter inquired. "Yes," his wife replied, "and just generally intolerable." Ferrucci told me he was recently prescribed a mouth guard because the stress of watching Watson play had him clenching his teeth excessively.

Ferrucci was never an aficionado of "Jeopardy!" ("I've certainly seen it," he said with a shrug. "I'm not a big fan.") But he craved an ambitious goal that would impel him to break new ground, that would verge on science fiction, and this fit the bill. "The computer on 'Star Trek' is a question-answering machine," he says. "It understands what you're asking and provides just the right chunk of response that you needed. When is the computer going to get to a point where the computer knows how to talk to you? That's my question."

What makes language so hard for computers, Ferrucci explained, is that it's full of "intended meaning." When people decode what someone else is saying, we can easily unpack the many nuanced allusions and connotations in every sentence. He gave me an example in the form of a "Jeopardy!" clue: "The name of this hat is elementary, my dear contestant." People readily detect the wordplay here — the echo of "elementary, my dear Watson," the famous phrase associated with *Sherlock Holmes* — and immediately recall that the Hollywood version of Holmes sports a deerstalker hat. But for a computer, there is no simple way to identify "elementary, my dear contestant" as wordplay. Cleverly matching different keywords, and even different fragments of the sentence — which in part is how most search engines work these days — isn't enough, either. (Type that clue into Google, and you'll get first-page referrals to "elementary, my dear watson" but none to deerstalker hats.)

What's more, even if a computer determines that the actual underlying question is "What sort of hat does Sherlock Holmes wear?" its data may not be stored in such a way that enables it to extract a precise answer. For years, computer scientists built question-answering systems by creating specialized databases, in which certain facts about the world were recorded and linked together. You could do this with Sherlock Holmes by building a database that includes connections between catchphrases and his hat and his violin-playing. But that database would be pretty narrow; it wouldn't be able to answer questions about nuclear power, or fish species, or the history of France. Those would require their own hand-made databases. Pretty soon you'd face the impossible task of organizing all the information known to man — of "boiling the ocean," as Ferrucci put it. In computer science, this is known as a "bottleneck" problem. And even if you could get past it, you might then face the issue of "brittleness": if your database contains only facts you input manually, it breaks any time you ask it a question about something beyond that

material. There's no way to hand-write a database that would include the answer to every "Jeopardy!" clue, because the subject matter is potentially all human knowledge.

The great shift in artificial intelligence began in the last 10 years, when computer scientists began using statistics to analyze huge piles of documents, like books and news stories. They wrote algorithms that could take any subject and automatically learn what types of words are, statistically speaking, most (and least) associated with it. Using this method, you could put hundreds of articles and books and movie reviews discussing Sherlock Holmes into the computer, and it would calculate that the words "deerstalker hat" and "Professor Moriarty" and "opium" are frequently correlated with one another, but not with, say, the [Super Bowl](#). So at that point you could present the computer with a question that didn't mention Sherlock Holmes by name, but if the machine detected certain associated words, it could conclude that Holmes was the probable subject — and it could also identify hundreds of other concepts and words that weren't present but that were likely to be related to Holmes, like "Baker Street" and "chemistry."

In theory, this sort of statistical computation has been possible for decades, but it was impractical. Computers weren't fast enough, memory wasn't expansive enough and in any case there was no easy way to put millions of documents into a computer. All that changed in the early '00s. Computer power became drastically cheaper, and the amount of online text exploded as millions of people wrote blogs and wikis about anything and everything; news organizations and academic journals also began putting all their works in digital format. What's more, question-answering experts spent the previous couple of decades creating several linguistic tools that helped computers puzzle through language — like rhyming dictionaries, bulky synonym finders and "classifiers" that recognized the parts of speech.

Still, the era's best question-answering systems remained nowhere near being able to take on "Jeopardy!" In 2006, Ferrucci tested I.B.M.'s most advanced system — it wasn't the best in its field but near the top — by giving it 500 questions from previous shows. The results were dismal. He showed me a chart, prepared by I.B.M., of how real-life "Jeopardy!" champions perform on the TV show. They are clustered at the top in what Ferrucci calls "the winner's cloud," which consists of individuals who are the first to hit the buzzer about 50 percent of the time and, after having "won" the buzz, solve on average 85 to 95 percent of the clues. In contrast, the I.B.M. system languished at the bottom of the chart. It was rarely confident enough to answer a question, and when it was, it got the right answer only 15 percent of the time. Humans were fast and smart; I.B.M.'s machine was slow and dumb.

"Humans are just — boom! — they're just plowing through this in just seconds," Ferrucci said excitedly. "They're getting the questions, they're breaking them down, they're interpreting them, they're getting the right interpretation, they're looking this up in their

memory, they're scoring, they're doing all this just instantly.”

But Ferrucci argued that I.B.M. could be the one to finally play “Jeopardy!” If the firm focused its computer firepower — including its new “BlueGene” servers — on the challenge, Ferrucci could conduct experiments dozens of times faster than anyone had before, allowing him to feed more information into Watson and test new algorithms more quickly. Ferrucci was ambitious for personal reasons too: if he didn’t try this, another computer scientist might — “and then bang, you are irrelevant,” he told me.

“I had no interest spending the next five years of my life pursuing things in the small,” he said. “I wanted to push the limits.” If they could succeed at “Jeopardy!” soon after that they could bring the underlying technology to market as customizable question-answering systems. In 2007, his bosses gave him three to five years and increased his team to 15 people.

FERRUCCI'S MAIN breakthrough was not the design of any single, brilliant new technique for analyzing language. Indeed, many of the statistical techniques Watson employs were already well known by computer scientists. One important thing that makes Watson so different is its enormous speed and memory. Taking advantage of I.B.M.’s supercomputing heft, Ferrucci’s team input millions of documents into Watson to build up its knowledge base — including, he says, “books, reference material, any sort of dictionary, thesauri, folksonomies, taxonomies, encyclopedias, any kind of reference material you can imagine getting your hands on or licensing. Novels, bibles, plays.”

Watson’s speed allows it to try thousands of ways of simultaneously tackling a “Jeopardy!” clue. Most question-answering systems rely on a handful of algorithms, but Ferrucci decided this was why those systems do not work very well: no single algorithm can simulate the human ability to parse language and facts. Instead, Watson uses more than a hundred algorithms at the same time to analyze a question in different ways, generating hundreds of possible solutions. Another set of algorithms ranks these answers according to plausibility; for example, if dozens of algorithms working in different directions all arrive at the same answer, it’s more likely to be the right one. In essence, Watson thinks in probabilities. It produces not one single “right” answer, but an enormous number of possibilities, then ranks them by assessing how likely each one is to answer the question.

Ferrucci showed me how Watson handled this sample “Jeopardy!” clue: “He was presidentially pardoned on Sept. 8, 1974.” In the first pass, the algorithms came up with “Nixon.” To evaluate whether “Nixon” was the best response, Watson performed a clever trick: it inserted the answer into the original phrase — “Nixon was presidentially pardoned on Sept. 8, 1974” — and then ran it as a new search, to see if it also produced results that supported “Nixon” as the right answer. (It did. The new search returned the result “Ford

pardoned Nixon on Sept. 8, 1974,” a phrasing so similar to the original clue that it helped make “Nixon” the top-ranked solution.)

Other times, Watson uses algorithms that can perform basic cross-checks against time or space to help detect which answer seems better. When the computer analyzed the clue “In 1594 he took a job as a tax collector in Andalusia,” the two most likely answers generated were “Thoreau” and “Cervantes.” Watson assessed “Thoreau” and discovered his birth year was 1817, at which point the computer ruled him out, because he wasn’t alive in 1594. “Cervantes” became the top-ranked choice.

When Watson is playing a game, Ferrucci lets the audience peek into the computer’s analysis. A monitor shows Watson’s top five answers to a question, with a bar graph beside each indicating its confidence. During one of my visits, the host read the clue “Thousands of prisoners in the Philippines re-enacted the moves of the video of this [Michael Jackson](#) hit.” On the monitor, I could see that Watson’s top pick was “Thriller,” with a confidence level of roughly 80 percent. This answer was correct, and Watson buzzed first, so it won \$800. Watson’s next four choices — “Music video,” “Billie Jean,” “Smooth Criminal” and “[MTV](#)” — had only slivers for their bar graphs. It was a fascinating glimpse into the machine’s workings, because you could spy the connective thread running between the possibilities, even the wrong ones. “Billie Jean” and “Smooth Criminal” were also major hits by Michael Jackson, and “MTV” was the main venue for his videos. But it’s very likely that none of those correlated well with “Philippines.”

After a year, Watson’s performance had moved halfway up to the “winner’s cloud.” By 2008, it had edged into the cloud; on paper, anyway, it could beat some of the lesser “Jeopardy!” champions. Confident they could actually compete on TV, I.B.M. executives called up Harry Friedman, the executive producer of “Jeopardy!” and raised the possibility of putting Watson on the air.

Friedman told me he and his fellow executives were surprised: nobody had ever suggested anything like this. But they quickly accepted the challenge. “Because it’s I.B.M., we took it seriously,” Friedman said. “They had the experience with Deep Blue and the chess match that became legendary.”

WHEN THEY FIRST showed up to play Watson, many of the contestants worried that they didn’t stand a chance. Human memory is frail. In a high-stakes game like “Jeopardy!” players can panic, becoming unable to recall facts they would otherwise remember without difficulty. Watson doesn’t have this problem. It might have trouble with its analysis or be unable to logically connect a relevant piece of text to a question. But it doesn’t forget things. Plus, it has lightning-fast reactions — wouldn’t it simply beat the humans to the buzzer every time?

"We're relying on nerves — old nerves," Dorothy Gilmartin complained, halfway through her first game, when it seemed that Watson was winning almost every buzz.

Yet the truth is, in more than 20 games I witnessed between Watson and former "Jeopardy!" players, humans frequently beat Watson to the buzzer. Their advantage lay in the way the game is set up. On "Jeopardy!" when a new clue is given, it pops up on screen visible to all. (Watson gets the text electronically at the same moment.) But contestants are not allowed to hit the buzzer until the host is finished reading the question aloud; on average, it takes the host about six or seven seconds to read the clue.

Players use this precious interval to figure out whether or not they have enough confidence in their answers to hazard hitting the buzzer. After all, buzzing carries a risk: someone who wins the buzz on a \$1,000 question but answers it incorrectly loses \$1,000.

Often those six or seven seconds weren't enough time for Watson. The humans reacted more quickly. For example, in one game an \$800 clue was "In Poland, pick up some *kalafjor* if you crave this broccoli relative." A human contestant jumped on the buzzer as soon as he could. Watson, meanwhile, was still processing. Its top five answers hadn't appeared on the screen yet. When these finally came up, I could see why it took so long. Something about the question had confused the computer, and its answers came with mere slivers of confidence. The top two were "vegetable" and "cabbage"; the correct answer — "cauliflower" — was the third guess.

To avoid losing money — Watson doesn't care about the money, obviously; winnings are simply a way for I.B.M. to see how fast and accurately its system is performing — Ferrucci's team has programmed Watson generally not to buzz until it arrives at an answer with a high confidence level. In this regard, Watson is actually at a disadvantage, because the best "Jeopardy!" players regularly hit the buzzer as soon as it's possible to do so, even if it's before they've figured out the clue. "Jeopardy!" rules give them five seconds to answer after winning the buzz. So long as they have a good feeling in their gut, they'll pounce on the buzzer, trusting that in those few extra seconds the answer will pop into their heads. Ferrucci told me that the best human contestants he had brought in to play against Watson were amazingly fast. "They can buzz in 10 milliseconds," he said, sounding astonished. "Zero milliseconds!"

On the third day I watched Watson play, it did quite poorly, losing four of seven games, in one case without any winnings at all. Often Watson appeared to misunderstand the clue and offered answers so inexplicable that the audience erupted in laughter. Faced with the clue "This 'insect' of a gangster was a real-life hit man for Murder Incorporated in the 1930s & '40s," Watson responded with "James Cagney." Up on the screen, I could see that none of its lesser choices were the correct one, "Bugsy Siegel." Later, when asked to complete the

phrase “Toto, I’ve a feeling we’re not in Ka—,” Watson offered “not in Kansas anymore,” which was incorrect, since the precise phrasing was simply “Kansas anymore,” and “Jeopardy!” is strict about phrasings. When I looked at the screen, I noticed that the answers Watson had ranked lower were pretty odd, including “Steve Porcaro,” the keyboardist for the band Toto (which made a vague sort of sense), and “[Jackie Chan](#)” (which really didn’t). In another game, Watson’s logic appeared to fall down some odd semantic rabbit hole, repeatedly giving the answer “[Tommy Lee Jones](#)” — the name of the Hollywood actor — to several clues that had nothing to do with him.

In the corner of the conference room, Ferrucci sat typing into a laptop. Whenever Watson got a question wrong, Ferrucci winced and stamped his feet in frustration, like a college-football coach watching dropped passes. “This is *torture*,” he added, laughing.

Seeing Watson’s errors, you can sometimes get a sense of its cognitive shortcomings. For example, in “Jeopardy!” the category heading often includes a bit of wordplay that explains how the clues are to be addressed. Watson sometimes appeared to mistakenly analyze the entire category and thus botch every clue in it. One game included the category “Stately Botanical Gardens,” which indicated that every clue would list several gardens, and the answer was the relevant state. Watson clearly didn’t grasp this; it answered “botanic garden” repeatedly. I also noticed that when Watson was faced with very short clues — ones with only a word or two — it often seemed to lose the race to the buzzer, possibly because the host read the clues so quickly that Watson didn’t have enough time to do its full calculations. The humans, in contrast, simply trusted their guts and jumped.

Ferrucci refused to talk on the record about Watson’s blind spots. He’s aware of them; indeed, his team does “error analysis” after each game, tracing how and why Watson messed up. But he is terrified that if competitors knew what types of questions Watson was bad at, they could prepare by boning up in specific areas. I.B.M. required all its sparring-match contestants to sign nondisclosure agreements prohibiting them from discussing their own observations on what, precisely, Watson was good and bad at. I signed no such agreement, so I was free to describe what I saw; but Ferrucci wasn’t about to make it easier for me by cataloguing Watson’s vulnerabilities.

Computer scientists I spoke to agreed that witty, allusive clues will probably be Watson’s weak point. “Retrieval of obscure Italian poets is easy — [Watson] will never forget that one,” Peter Norvig, the director of research at Google, told me. “But ‘Jeopardy!’ tends to have a lot of wordplay, and that’s going to be a challenge.” Certainly on many occasions this seemed to be true. Still, at other times I was startled by Watson’s eerily humanlike ability to untangle astonishingly coy clues. During one game, a category was “All-Eddie Before & After,” indicating that the clue would hint at two different things that need to be blended together, one of which included the name “Eddie.” The \$2,000 clue was “A ‘Green Acres’

star goes existential (& French) as the author of 'The Fall.' " Watson nailed it perfectly: "Who is Eddie Albert Camus?"

Ultimately, Watson's greatest edge at "Jeopardy!" probably isn't its perfect memory or lightning speed. It is the computer's lack of emotion. "Managing your emotions is an enormous part of doing well" on "Jeopardy!" Bob Harris, a five-time champion, told me. "Every single time I've ever missed a Daily Double, I always miss the next clue, because I'm still kicking myself." Because there is only a short period before the next clue comes along, the stress can carry over. Similarly, humans can become much more intimidated by a \$2,000 clue than a \$200 one, because the more expensive clues are presumably written to be much harder.

Whether Watson will win when it goes on TV in a real "Jeopardy!" match depends on whom "Jeopardy!" pits against the computer. Watson will not appear as a contestant on the regular show; instead, "Jeopardy!" will hold a special match pitting Watson against one or more famous winners from the past. If the contest includes Ken Jennings — the best player in "Jeopardy!" history, who won 74 games in a row in 2004 — Watson will lose if its performance doesn't improve. It's pretty far up in the winner's cloud, but it's not yet at Jennings's level; in the sparring matches, Watson was beaten several times by opponents who did nowhere near as well as Jennings. (Indeed, it sometimes lost to people who hadn't placed first in their own appearances on the show.) The show's executive producer, Harry Friedman, will not say whom it is picking to play against Watson, but he refused to let Jennings be interviewed for this story, which is suggestive.

Ferrucci says his team will continue to fine-tune Watson, but improving its performance is getting harder. "When we first started, we'd add a new algorithm and it would improve the performance by 10 percent, 15 percent," he says. "Now it'll be like half a percent is a good improvement."

Ferrucci's attitude toward winning is conflicted. I could see that he hungers to win. And losing badly on national TV might mean negative publicity for I.B.M. But Ferrucci also argued that Watson might lose merely because of bad luck. Should one of Watson's opponents land on both Daily Doubles, for example, that player might double his or her money and vault beyond Watson's ability to catch up, even if the computer never flubs another question.

Ultimately, Ferrucci claimed not to worry about winning or losing. He told me he's happy that I.B.M. has simply pushed this far and produced a system that performs so well at answering questions. Even a televised flameout, he said, won't diminish the street cred Watson will give I.B.M. in the computer-science field. "I don't really care about 'Jeopardy!'" he told me, shrugging.

I.B.M. PLANS TO begin selling versions of Watson to companies in the next year or two. John Kelly, the head of I.B.M.'s research labs, says that Watson could help decision-makers sift through enormous piles of written material in seconds. Kelly says that its speed and quality could make it part of rapid-fire decision-making, with users talking to Watson to guide their thinking process.

"I want to create a medical version of this," he adds. "A Watson M.D., if you will." He imagines a hospital feeding Watson every new medical paper in existence, then having it answer questions during split-second emergency-room crises. "The problem right now is the procedures, the new procedures, the new medicines, the new capability is being generated faster than physicians can absorb on the front lines and it can be deployed." He also envisions using Watson to produce virtual call centers, where the computer would talk directly to the customer and generally be the first line of defense, because, "as you've seen, this thing can answer a question faster and more accurately than most human beings."

"I want to create something that I can take into every other retail industry, in the transportation industry, you name it, the banking industry," Kelly goes on to say. "Any place where time is critical and you need to get advanced state-of-the-art information to the front of decision-makers. Computers need to go from just being back-office calculating machines to improving the intelligence of people making decisions." At first, a Watson system could cost several million dollars, because it needs to run on at least one \$1 million I.B.M. server. But Kelly predicts that within 10 years an artificial brain like Watson could run on a much cheaper server, affordable by any small firm, and a few years after that, on a laptop.

Ted Senator, a vice president of SAIC — a high-tech firm that frequently helps design government systems — is a former "Jeopardy!" champion and has followed Watson's development closely; in October he visited I.B.M. and played against Watson himself. (He lost.) He says that Watson-level artificial intelligence could make it significantly easier for citizens to get answers quickly from massive, ponderous bureaucracies. He points to the recent "[cash for clunkers](#)" program. He tried to participate, but when he went to the government site to see if his car qualified, he couldn't figure it out: his model, a 1995 Saab 9000, was listed twice, each time with different mileage-per-gallon statistics. What he needed was probably buried deep inside some government database, but the bureaucrats hadn't presented the information clearly enough. "So I gave up," he says. This is precisely the sort of task a Watson-like artificial intelligence can assist in, he says. "You can imagine if I'm applying for health insurance, having to explain the details of my personal situation, or if I'm trying to figure out if I'm eligible for a particular tax deduction. Any place there's massive data that surpasses the human's ability to sort through it, and there's a time constraint on getting an answer."

Many experts imagine even quirkier ways that everyday life might be transformed as

question-answering technology becomes more powerful and widespread. Andrew Hickl, the C.E.O. of Language Computer Corporation, which makes question-answering systems, among other things, for businesses, was recently asked by a client to make a “contradiction engine”: if you tell it a statement, it tries to find evidence on the Web that contradicts it. ‘It’s like, ‘I believe that Dallas is the most beautiful city in the United States,’ and I want to find all the evidence on the Web that contradicts that.’” (It produced results that were only 70 percent relevant, which satisfied his client.) Hickl imagines people using this sort of tool to read through the daily news. “We could take something that Harry Reid says and immediately figure out what contradicts it. Or somebody tweets something that’s wrong, and we could automatically post a tweet saying, ‘No, actually, that’s wrong, and here’s proof.’”

CULTURALLY, OF COURSE, advances like Watson are bound to provoke nervous concerns too. High-tech critics have begun to wonder about the wisdom of relying on artificial-intelligence systems in the face of complex reality. Many Wall Street firms, for example, now rely on “millisecond trading” computers, which detect deviations in prices and order trades far faster than humans ever could; but these are now regarded as a possible culprit in the seemingly irrational hourlong stock-market plunge of the spring. Would doctors in an E.R. feel comfortable taking action based on a split-second factual answer from a Watson M.D.? And while service companies can clearly save money by relying more on question-answering systems, they are precisely the sort of labor-saving advance deplored by unions — and customers who crave the ability to talk to a real, intelligent human on the phone.

Some scientists, moreover, argue that Watson has serious limitations that could hamper its ability to grapple with the real world. It can analyze texts and draw basic conclusions from the facts it finds, like figuring out if one event happened later than another. But many questions we want answered require more complex forms of analysis. Last year, the computer scientist Stephen Wolfram released “Wolfram Alpha,” a question-answering engine that can do mathematical calculations about the real world. Ask it to “compare the populations of New York City and Cincinnati,” for example, and it will not only give you their populations — 8.4 million versus 333,336 — it will also create a bar graph comparing them visually and calculate their ratio (25.09 to 1) and the percentage relationship between them (New York is 2,409 percent larger). But this sort of automated calculation is only possible because Wolfram and his team spent years painstakingly hand-crafting databases in a fashion that enables a computer to perform this sort of analysis — by typing in the populations of New York and Cincinnati, for example, and tagging them both as “cities” so that the engine can compare them. This, Wolfram says, is the deep challenge of artificial intelligence: a lot of human knowledge isn’t represented in words alone, and a computer won’t learn that stuff just by encoding English language texts, as Watson does. The only

way to program a computer to do this type of mathematical reasoning might be to do precisely what Ferrucci doesn't want to do — sit down and slowly teach it about the world, one fact at a time.

"Not to take anything away from this 'Jeopardy!' thing, but I don't think Watson really is answering questions — it's not like the 'Star Trek' computer," Wolfram says. (Of course, Wolfram Alpha cannot answer the sort of broad-ranging trivia questions that Watson can, either, because Wolfram didn't design it for that purpose.) What's more, Watson can answer only questions asking for an objectively knowable fact. It cannot produce an answer that requires judgment. It cannot offer a new, unique answer to questions like "What's the best high-tech company to invest in?" or "When will there be peace in the Middle East?" All it will do is look for source material in its database that appears to have addressed those issues and then collate and compose a string of text that seems to be a statistically likely answer. Neither Watson nor Wolfram Alpha, in other words, comes close to replicating human wisdom.

At best, Ferrucci suspects that Watson might be simulating, in a stripped-down fashion, some of the ways that our human brains process language. Modern neuroscience has found that our brain is highly "parallel": it uses many different parts simultaneously, harnessing billions of neurons whenever we talk or listen to words. "I'm no cognitive scientist, so this is just speculation," Ferrucci says, but Watson's approach — tackling a question in thousands of different ways — may succeed precisely because it mimics the same approach. Watson doesn't come up with an answer to a question so much as make an educated guess, based on similarities to things it has been exposed to. "I have young children, you can see them guessing at the meaning of words, you can see them guessing at grammatical structure," he notes.

This is why Watson often seemed most human not when it was performing flawlessly but when it wasn't. Many of the human opponents found the computer most endearing when it was clearly misfiring — misinterpreting the clue, making weird mistakes, rather as we do when we're put on the spot.

During one game, the category was, coincidentally, "I.B.M." The questions seemed like no-brainers for the computer (for example, "Though it's gone beyond the corporate world, I.B.M. stands for this" — "International Business Machines"). But for some reason, Watson performed poorly. It came up with answers that were wrong or in which it had little confidence. The audience, composed mostly of I.B.M. employees who had come to watch the action, seemed mesmerized by the spectacle.

Then came the final, \$2,000 clue in the category: "It's the last name of father and son Thomas Sr. and Jr., who led I.B.M. for more than 50 years." This time the computer

pounced. "Who is Watson?" it declared in its synthesized voice, and the crowd erupted in cheers. At least it knew its own name.

Clive Thompson, a contributing writer for the magazine, writes frequently about technology and science.

Computers Solve Checkers—It's a Draw

King me! Top computer scientist proves perfect play leads to draw, recounts battle for world championship, gets kinged

By [JR Minkel](#)



CHECKERS IS SOLVED: Eighteen years of number crunching concludes that perfect checkers ends in a draw. Image: © ISTOCKPHOTO/CHRISTOPHER O DRISCOLL



The Best Science Writing Online 2012

Showcasing more than fifty of the most provocative, original, and significant online essays from 2011, The Best Science Writing Online 2012 will change the way...

Read More »

Jonathan Schaeffer's quest for the perfect game of checkers has ended. The 50-year-old computer scientist from the University of Alberta in Edmonton left human players in the dust more than a decade ago after a trial by fire against the greatest checkers champion in history.

And now, after putting dozens of computers to work night and day for 18 years—*jump, jump, jump*—he says he has solved the game—*king me!*. "The starting position, assuming no side makes a mistake, is a draw," he says.

Schaeffer's proof, described today in *Science* (and freely [available here for others to verify](#)), would make checkers the most complex game yet solved by machines, beating out the checker-stacking game Connect Four in difficulty by a factor of a million.

"It's a milestone," says Murray Campbell, a computer scientist at IBM's T. J. Watson Research Center in Hawthorne, N.Y., and co-inventor of the chess program Deep Blue. "He's stretched the state of the art."

Although technological limits prohibit analyzing each of the 500 *billion billion* possible arrangements that may appear on an eight-by-eight checkerboard, Schaeffer and his team identified moves that guaranteed the game would end in a draw no matter how tough the competition.

Like any complicated mathematical proof, the result will have to withstand scrutiny. But "it's close to 100 percent," says computer scientist Jaap van den Herik of Maastricht University in the Netherlands, who has seen the details. "He has never published anything that was not completely true."

Opening Play: Walking a Precipice

Schaeffer's odyssey began in the late 1980s. He had written a top chess program but IBM was on the verge of pouring its far vaster resources into Deep Blue. "I like to be competitive," he says, so he turned his attention elsewhere. "I naively thought I could solve the game of checkers," he recalls. "You can teach somebody the rules in a minute."

Setting out in 1989 with 16 megabytes of computer memory, he quickly found that checkers, like chess, was too rich with possible positions to dash off a solution. So he switched gears, vowing to topple legendary checkers champion Marion Tinsley, who had lost only three games in tournament play since 1950.

In 1992 Schaeffer's program Chinook [took on Tinsley](#), who had resigned as world champion when the American Checker Federation and English Draughts Association temporarily refused to sanction the man-computer matchup.

Tinsley was so good that his opponents played dull games in the hope of securing at least a draw, according to Schaeffer; Chinook apparently put the magic back in the game for the champ. "It played brash, aggressive moves—it walked on the edge of a precipice," Scheaffer anthropomorphizes. "It would do things people looked at and said, 'Man, is that program crazy?'"

The program actually beat Tinsley twice, but computer glitches led to a forfeit that gave the human a 3–2 lead with two games left in a best-of-40 match. Schaeffer set Chinook on an aggressive course to try to recoup, resulting in another loss for the computer that cost it and its creator the match, Schaeffer recounted in his book *One Jump Ahead*.

In a rematch two years later, Tinsley withdrew suddenly after six drawn games, citing health problems, making Chinook the winner of the Man–Machine World Championship by default. The checkers champ was diagnosed with pancreatic [cancer](#), which killed him less than a year later.

Chinook soundly beat the top human player in 1996, but Schaeffer's memories of those years are bittersweet. "With hindsight, maybe winning was more important to me at that time than other things. I was a bit obsessed," he says. "My wife would say more than a bit obsessed."

Endgame: King Me!

Early this decade he decided that improved computer speed and memory had brought his goal of solving the game within reach. The first of the solution's two pieces builds on work begun during the Tinsley days. The Alberta researchers exhaustively checked the final stages of play for any arrangement of 10 pieces or less, identifying which of the 3.9×10^{13} positions won, lost or drew.

Next they identified 19 representative opening sequences and searched through subsequent moves for the easiest-to-find connections to final positions—win, lose or draw. To save time, they ignored pointless back-and-forth moves or those that did not turn a draw into a win.

If the sequence of possible moves branches like a tree, the group found that main branch after main branch led to a draw. On April 27, Schaeffer says, the program traced the final branches, completing his 18-year task. He had received his king.

The result itself was not surprising. "We expected that it should be a draw, because at the highest levels there are many games that end in a draw," Herik says.

What the proof does, Watson's Campbell says, is highlight the power of methodically checking outcomes one by one. The most robust chess programs such as Deep Blue and Deep Fritz have beaten world champions by combining swift number crunching with rules of thumb derived from years of human play.

The checkers solver is comparatively dumb, Campbell notes. "All it knows is this move wins or this move doesn't," he says. "It can't 'explain' what it knows."

For that reason, Herik says the results won't be much help to players until researchers can figure out how to translate them into rules or procedures the human mind can follow—a major challenge in itself.

Next Move?

So what game will fall to computers next? Schaeffer and colleagues speculate it will be Othello, an eight-by-eight disk-flipping game. Chess presents a far mightier challenge, but researchers are "in the realm of thinking about" solving it, Herik says, which he calls "a tremendous achievement."

The great white whale of games remains the Asian pastime Go. Although programs have [recently become competitive with grand masters](#) on nine-by-nine boards, they remain toothless on the full 19-by-19 game.

Schaeffer, a true game lover—his two dogs are named Scrabble and Rummicub—intends to pit [his poker program](#) against two top players next week in a 500-hand match designed to minimize the role of chance.

He has come a long way since being fixated on the world championship. Whatever games lie ahead, he can take satisfaction in his current victory. "Eighteen years later," he says, "I'm finally done."

Forbes



Chunka Mui, Contributor
I help design and stress-test innovation strategies

LEADERSHIP | 1/22/2013 @ 10:12AM | 321,493 views

Fasten Your Seatbelts: Google's Driverless Car Is Worth Trillions (Part 1)

Part One of a Seven-Part Series

Much of the reporting about Google's driverless car has mistakenly focused on its science-fiction feel. While the car is certainly cool—just watch the video



below about a 95%-blind man running errands—the gee-whiz focus suggests that it is just a high-tech dalliance by a couple of brash young multibillionaires, Google founders Larry Page and Sergey Brin.

In fact, the driverless car has broad implications for society, for the economy and for individual businesses. Just in the U.S., the car puts up for grab some **\$2 trillion a year in revenue and even more market cap.** It creates business opportunities that dwarf Google's current search-based business and unleashes existential challenges to market leaders across numerous industries, including car makers, auto insurers, energy companies and others that share in car-related revenue.

Because people consistently underestimate the implications of a change in technology—are you listening, Kodak, Blockbuster, Borders, Sears, etc.?—and because many industries face the kind of disruption that may beset the auto industry, I'm going to do a series of blogs on the ripple effects that the driverless car may create. I'm hoping both to dramatize the effects of a disruptive technology and to illustrate how to think about the dangers and the opportunities that one creates.

In this installment, I'll start the series with a broad-brush look at the far-reaching changes that could occur from the driver's standpoint. In the next installment, I'll show just how far the ripples will reach for companies—not just car makers, but insurers, hospitals, parking lot operators and even governments and utilities. (Fines drop when every car obeys the law, and roads don't need to be lit if cars can see in the dark).

After that, I'll explore how real the prospects are for driverless cars. (Hint: The issue is when, not if—and when is sooner than you think.) In the last three installments, I'll go into the strategic implications for Google, for car makers and, finally, for every company thinking about innovation in these fast-moving times.

To begin:

Driverless car technology has the very real potential to save millions from death and injury and eliminate hundreds of billions of dollars of costs. Google's claims for the car, [as described by Sebastian Thrun](#), its lead developer, are:

1. We can reduce traffic accidents by 90%.
2. We can reduce wasted commute time and energy by 90%.
3. We can reduce the number of cars by 90%.

To put those claims in context:

About [5.5 million motor vehicle accidents](#) occurred in 2009 in the U.S., involving 9.5 million vehicles. These accidents killed 33,808 people and injured more than 2.2 million others, 240,000 of whom had to be hospitalized.

Adding up all costs related to accidents—including medical costs, property damage, loss of productivity, legal costs, travel delays and pain and lost quality of life—the American Automobile Association studied crash data in the 99 largest U.S. urban areas and [estimated the total costs to be \\$299.5 billion](#). Adjusting those numbers to cover the entire country suggests annual costs of about \$450 billion.

Now take 90% off these numbers. Google is claiming its car could save almost 30,000 lives each year on U.S. highways and prevent nearly 2 million additional injuries. Google claims it can reduce accident-related expenses by at least \$400 billion a year in the U.S. Even if Google is way off—and I don't believe it is—the improvement in safety will be startling.

In addition, the driverless car would reduce wasted commute

Great Mileage
Some Benefits of the Driverless Car

time and energy by relieving congestion and allowing cars to go faster, operate closer together and choose more effective routes.

[One study](#) estimated that traffic congestion wasted 4.8 billion hours and 1.9 billion gallons of fuel a year for urban Americans. That translates to \$101 billion in lost productivity and added fuel costs.

The driverless car could reduce the need for cars by enabling efficient sharing of vehicles. A driverless vehicle could theoretically be shared by multiple people, delivering itself when and where it is needed, parking itself in some remote place whenever it's not in use.

A car is often a person's second largest capital expenditure, after a home, yet a car sits unused some 95% of the time. With the Google car, people could avoid the outlay of many thousands of dollars, or tens of thousands, on an item that mostly sits and, instead, simply pay by the mile.

A [study](#) led by [Lawrence Burns](#) and William Jordon at Columbia University's Earth Institute [Program on Sustainable Mobility](#) showed the dramatic cost savings potential. Their analysis found that a shared, driverless fleet could provide far better mobility experiences than personally owned vehicles at far radically lower cost. For medium-sized cities like Ann Arbor, MI, the cost per trip-mile could be reduced by 80% when compared to personally own vehicles driven about 10,000 miles per year—without even factoring in parking and the opportunity cost of driving time. Their analysis showed similar cost savings potential for suburban and high-density urban scenarios as well.

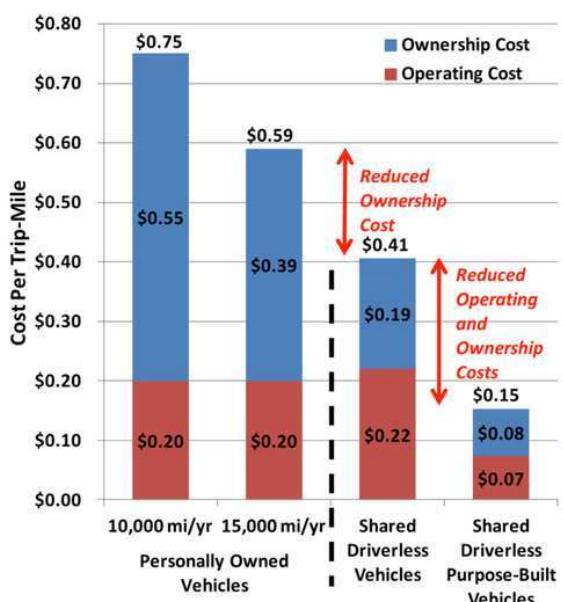
SOME BENEFITS OF THE DRIVERLESS CAR

Google's Aspiration	Potential Annual Benefits (US only)
• 90% reduction in accidents	<ul style="list-style-type: none"> • 4.95 million fewer accidents • 30,000 fewer deaths • 2 million fewer injuries • \$400 billion in accident-related cost savings
• 90% reduction wasted commuting	<ul style="list-style-type: none"> • 4.8 billion fewer commuting hours • 1.9 billion gallons in fuel savings • \$101 billion saved in lost productivity and fuel costs
• 90% reduction in cars	<ul style="list-style-type: none"> • Reduce cost per trip-mile by 80% or more • Increase car utilization from 5-10% to 75% or more • Better land use

Sources: Google, US NHTSA, AAA, Texas A&M Transportation Institute, Columbia University Earth Institute and Devil's Advocate Group's analysis

Shared Driverless Fleets

Personal travel costs can be dramatically reduced



Source: Program on Sustainable Mobility,
The Earth Institute, Columbia University

Driving could
become Zipcar writ large (except the car comes to you).

Looking worldwide, the statistics are less precise, but the potential benefits are even more startling. The World Health Organization [estimates](#) that more than 1.2 million people are killed on the world's roads each year, and as many as 50 million others are injured. And the WHO predicts that the problems will only get worse. It estimates that road traffic injuries will become the fifth leading cause of worldwide death by 2030, accounting for 3.6% of the total—rising from the ninth leading cause in 2004, when it accounted for 2.2% of the world total.

If Google could give everyone a world-class electronic driver, it would drastically reduce the deaths, injuries and direct costs of accidents. The driverless car might also save developing countries from ever having to replicate the car-centric infrastructure that has emerged in most western countries. This leapfrogging has already happened with telephone systems: Developing countries that lacked land-line telephone and broadband connectivity, [such as India](#), made the leap directly to mobile systems rather than build out their land-line infrastructures.

China alone expects to [invest almost \\$800 billion on road and highway construction](#) between 2011 and 2015. It is doubtful, however, whether even this massive investment can keep up with the rising accidents and traffic congestion that the country endures. And road construction won't deal with the issue of pollution, to which the massive car buildup contributes and which, as the following FT video reports, is becoming an ever more politically sensitive issue.

How might China and other developing economic powers' massive car-related investments be redeployed if fundamental assumptions were viewed through the lens of the driverless car?

In sum, the Google driverless car not only makes for a great demo; it has worldwide social and economic benefits that could amount to trillions of dollars per year.

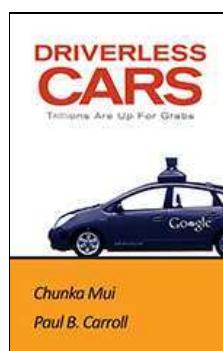
* * *

There is an old saying: One man's savings are another man's lost revenue. In [Part 2 of this series](#), I explore the enormous business threats and opportunities that will stem from Google's driverless car.

— — —

This series draws from research for [The New Killer Apps: This Time, Incumbents Can Beat Startups](#), a forthcoming book coauthored with Paul B. Carroll. To learn more, visit the [book's Facebook page](#).

An ebook, [Driverless Cars: Trillions Are Up For Grabs](#), which integrates the seven columns of this series, incorporates many of the comments and adds our latest research, is available at [Amazon](#).



Forbes Series Outline:

1. [Fasten Your Seatbelts: Google's Driverless Car Is Worth Trillions](#)
2. [The Ripple Effects—As Far As The Eye Can See](#)
3. [Why Change Will Come Sooner Than You Think](#)
4. [How Google Wins](#)
5. [How Automakers Still Win](#)
6. [Will Auto Insurers Survive Their Collision with Driverless Cars?](#)
7. [Driverless Cars Are Just One of Many Looming Disruptions](#)

Follow Chunka Mui here at Forbes (click on the +follow button next to his picture), on Twitter [@chunkamui](#) or at [Google+](#).

This article is available online at:

<http://www.forbes.com/sites/chunkamui/2013/01/22/fasten-your-seatbelts-googles-driverless-car-is-worth-trillions/>

Curriculum (/bjc-course/curriculum) / Unit 4 (/bjc-course/curriculum/04-conditionals-and-ai) /

Lab 1 (/bjc-course/curriculum/04-conditionals-and-ai/labs/01-random-numbers-and-conditionals) /

Lab: Random Numbers and Conditionals

In this lab you will create a guessing game. Look at the script below. The computer chooses a random number and then asks the player to guess the random number.

Algorithm

```
When flag clicked
  Set the secret number 1 to 10
  Repeat forever
    Ask for guess and wait
    If the answer = the secret number
      Say Guessed It Message
      Stop the loop
```

Script



Step 0

Before beginning to build your program, let's look at what variables we will need to begin the game.

- The secret number
- The guess from the player

Next, we need to figure out the tasks that the sprite and player will be enacting.

- The Sprite will begin by setting up the game's variables and choosing the secret number.

- The Sprite will ask for a guess from the player.
- The player will input their guess.
- If the player's guess was correct, the sprite will give the player a message and the game will end.

Note that there is a new block above that will return a random number between the two input numbers (inclusive). This block is in the Operator's palette. You can enter a number or use a variable. Because this is a reporter block you can use it inside other blocks as an input.



The game is not that cool yet, though. In the next few steps, we are going to make the game much more like playing the number-guessing game with a person! Here are tasks to try to make the code act more like a human! If you think of anything else that would make it cooler, feel free to add it in. Try each of these in order. When you are done, your finished code should be able to deal with all of the tasks.

Step 1

Add to your beginning script the blocks so that the sprite welcomes the player and asks for their name before beginning.



Step 2

We want to give the player more information if they don't guess correctly. Have the sprite tell the player if the secret number is bigger or smaller than the number that they guessed.

Step 3

Right now, the sprite always picks a number between 1 and 10. Change this so that the sprite always picks a number between 1 and a variable named max. (Don't forget to add the variable.) Add to your script so that it asks the player what they would like the maximum number to be, before choosing a random number. Use this maximum number as the highest number that the sprite will choose.

Step 4

Now let's keep track of how many guesses it takes before the player guesses the right number. You will need a new variable for this task and will need to add to it every time a guess occurs.

Step 5

When the player guesses the secret number, tell them how many guesses it took, and congratulate them using their name.

Unit 5: What's in a List

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts
- 3: The student can use computing tools and techniques for creative expression.
- 16: The student can express an algorithm in a language.
- 21: The student can evaluate a program for correctness.
- 22: The student can develop a correct program.
- 23: The student can employ appropriate mathematical and logical concepts in programming.
- 28: The student can analyze how computing affects communication, interaction, and cognition.
- 30: The student can analyze the beneficial and harmful effects of computing.
- 31: The student can connect computing within economic, social, and cultural contexts.

Readings/Lectures

- Lecture Slides 5.01: Lists ([/bjc-course/curriculum/05-lists/readings/01-lists-slides.pdf](#))

Labs/Exercises

- Lab 5.01: Lab 4.01 with Lists ([/bjc-course/curriculum/05-lists/labs/01-guessing-game-with-lists](#))
-

Introduction to Lists

Computer Science Principles

What are Lists?

- Lists are a type of data structure.

- This is a way to hold similar information in one 'container'.

- This is very useful as it allows us to store information such as a list of player's names.



List Blocks

Block	Purpose
Make a variable	Will create a variable that can become a list.
list []	Reports a newly created list with the given items. Use arrows to change the number of items. It is not required to be used with a variable as it is not required to create a named list in SNAP. A list not using a variable is called an "anonymous" list.

List Blocks

Block	Purpose
[in front of] v	Reports a new list that extends a list with a new item.

List Blocks

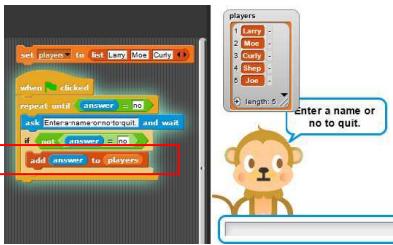
Block	Purpose
item [] of []	Reports an item from the list. You have the option of entering a number or numeric variable or choosing the 1, last, or any options.
all but first of []	Reports all but the first value in a list.
length of []	Reports the length of the list
[contains] [] [thing]	Reports true or false; True if the list contains the value; False if not.

List Blocks

Block	Purpose
add thing to []	Adds the value to the end of the list.
delete 1 of []	Deletes the value in the position of the list. You can choose 1, last, or all or use a numeric value or variable.
insert thing at 1 of []	Inserts the value at the position of the list. You can choose 1, last, or all or use a numeric value or variable.
replace item 1 of [] with []	Replaces the value at the position of the list with the given value. You can choose 1, last, or all or use a numeric value or variable.

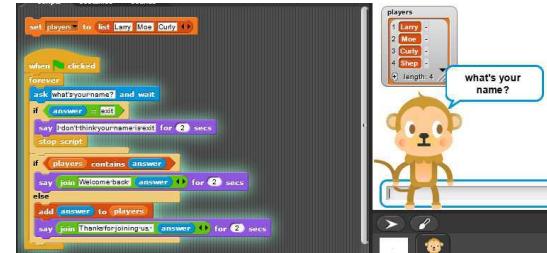
Adding to the List

- Use the `add () to []` block to add names (strings) to a list called `Players`.
 - Note I have a loop to continuously ask for a name with an option of entering “no” to stop the loop.
 - After entering 4 names, then “no”, my List contents are as shown.



Reading from the List

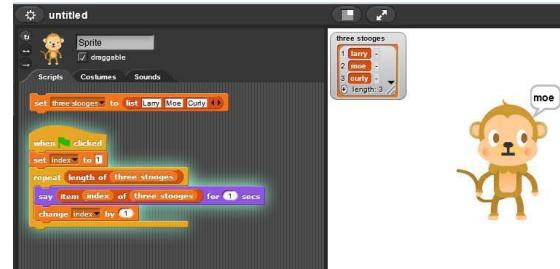
- The following script uses the `[] contains ()` block to see if the input is already in the list.
- If it is not in the `Players` list, the input is added to the list.



Looping Through a List

- You can loop through a list in order to read all values in the list.
- Note in the example script, there is a variable called `index`.
 - This variable will represent a value’s position in the list.
- **List Blocks used**
 - `length of []` returns the number of items in the list.
 - `Item () of []` will pull the value from the list at the position indicated by the `index` variable.

Looping Through a List



Curriculum (/bjc-course/curriculum) / Unit 5 (/bjc-course/curriculum/05-lists) /
Lab 1 (/bjc-course/curriculum/05-lists/labs/01-guessing-game-with-lists) /

Lab: Guessing Game version 2.0

The Number Guessing Game that you created last week had you utilize IF statements and random number generators. This week, you will expand upon that game and make it smarter! The goal of this Expanded Guessing Game (we'll call it, version 2.0) is to make sprites that play the game smarter! In addition, since this will make the game much faster, we'll make the guessing sprite have to get three guesses right to win.

Open your Guessing Game project from the last unit to use as a reference. Create a new project called `yourLastName_GuessingGame2`. This version will use two sprites: one to hold the secret number and give hints and the other to try to guess the secret.

Step 0

Before beginning to build your program, let's look at what variables we will need and what each sprite should do to play the game.

Variables:

- The guess from the guesser sprite
- List to hold the guesses
- The number of times the guesser sprite has guessed
- The secret number
- The number of times the guessing sprite wins

Tasks: * The First Sprite (the guide) needs to initialize the variables by setting them to 0 and picking a secret number.

- The First Sprite asks the second Sprite to guess.
- The Second Sprite (the guesser) guesses a number.
- The Second Sprite adds its guess to a list.
- The First Sprite gives a hint based on the guess.
- Repeat the second through fifth step until the guess is correct.
- Repeat the program until the second sprite wins three times.

Step 1

You will need to set up your 'Game Guide' sprite (or sprite 1) that will be asking for the guess just as it did in your last lab. Review the task list above if needed. The sprite will need to first set up all variables (don't forget to initialize) then repeatedly ask for a guess checking to see if the secret number is guessed or if a hint should be displayed. When the secret number is guessed, you want to stop and tell the number of guesses it took to win that game (and maybe say "Congrats" or another message). Go ahead and set up this part of your script. Don't forget you will need to 'ask' the guessing sprite for its guess. What is a good way to do this? How have you communicated with another sprite earlier?

Step 2

Let's create a Give Hint procedure and move the blocks that handle this task into the blocks editor. Now just add the Give Hint procedure into your sprite 1's script where it should be.

Step 3

Now let's work with the guessing sprite to add a list of the guesses. Remember a list is a collection that can contain multiple values of information. Think of a list as a container for multiple values. You can add, search for, and remove information from a list. Lists are very powerful data structures, and have many uses in Computer Science! For our lab, we'll make a simple list to store the guesses that our sprite has made so far.

To create a list, click "Make a Variable" in the Variables Tab.



Call your list variable whatever you like. The name should be something descriptive and informative though. In the example above, we named the list "GuessesList".

Now we are going to tell BYOB/SNAP that the variable is a list. Notice that when you click on the set command, you will see the variable show a list as its contents (it will not be a list until you click on the set command).



Step 4

Let's make a procedure called Guess Number for the guessing sprite to guess the number. To do that select the guessing sprite and click Make a Block in the Variables palette. Let's look at the algorithm for this new block. We want the sprite to guess a number between 1 and 10 (inclusive) and say the number.

Will you need a variable to hold the guess?

Algorithm (Pseudocode)

```
To Guess Number:  
Set the guess to a random number between 1 and 10, inclusive  
Say the guess.
```

Code Script in Blocks Editor



Add the code into the Block Editor. Click the block in the sprite's window to make certain it works as expected.

Step 5

Now let's expand this procedure so that it will keep track of the guesses this sprite has already guessed so that the sprite does not guess the same number twice. In order to do this task, we will want to keep a list of all the guesses (myGuess).

First, drag the `add (thing) to ()` block into your block editor. The “add ‘thing’ to...” block will add the myGuess value to your list. This is a very useful block that you’ll likely use in the future as well (hint hint). The “thing” can be replaced by either something you specify, or a variable. In this case, we want to add the number that we’ve just guessed to the list. Drag the variable “myGuess” into the Block Editor and snap it into the ‘thing’ field. It should look like this now:

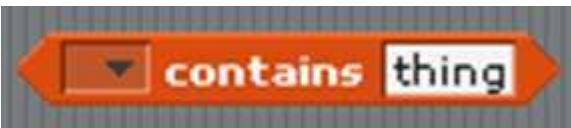


So now our block will pick a random number, say it, and add it to a list. This doesn’t quite yet accomplish what we need. What we want is for the “Guess Number” function to only choose a number as long as it hasn’t already been guessed. What we need here is a loop with an IF statement! You will need to add this in Try It! #1.

Try It! Guessing Game v. 2

Try It #1

Expand the “Guess Number” function in the guessing sprite so that it only says the guess and adds it to the list as long as the value of myGuess is not already in the list. This block with an IF block may come in handy.



Try It #2

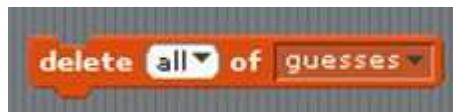
Next, let’s expand the script we have in the original sprite (not the guessing sprite). Add an additional loop around the game so that the “repeat until” loop keeps running until the guessing sprite has guessed the secret number correctly (or won) 3 times. You may need a wins variable for this (hint).

Run your game. Make sure the check mark beside your guesses list is checked, and you can see the list of guesses

populate itself on the stage as the guessing sprite adds guesses to its list! Notice that as guesses fills up, it takes the guessing sprite a bit longer to guess. This is because the guessing sprite is making sure that he does not guess a number he's already guessed! Unless your guessing sprite is very lucky, once guesses fills up, the guessing sprite will freeze and stop guessing. Why is this?

Fix the game so that the guessing sprite doesn't freeze! The way we have it now, once the guessing sprite wins, the guesses list is NOT cleared. The sprite should reset the guess list after winning! In addition, you'll notice that guesses does not reset when you restart the game. The list should also be cleared when the game starts. On top of that, all of the variables should be reset when the game starts!

This block should help:



Try It #3

Let's make the guessing sprite smarter! If you were playing a guessing game with someone and had guessed 3 and was told it was too low, would you guess 3 or lower? Of course not, you would guess 4 or above. Instead of guessing between 1 and 10, now you want the guessing sprite to guess between two values that will vary as guesses are made. If you want values to vary, maybe we should use variables to determine the low and the high for our guessing sprite to guess between.

So, give the same logic to the guessing sprite.

Try It #4

Test your game!



Unit 6: Where did I put it?

Searching and Sorting Algorithm

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 4: The student can use programming as a creative tool.
- 15: The student can develop an algorithm.
- 16: The student can express an algorithm in a language.
- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Readings/Lectures

- Blown to Bits: Chapter 4 (<http://www.bitsbook.com/wp-content/uploads/2008/12/chapter4.pdf>)
- Reading 6.01: All Algorithms are Not Equal (/bjc-course/curriculum/06-searching-sorting/readings/01-all-algorithms-are-not-equal)
- Reading 6.02: Algorithm Analysis (/bjc-course/curriculum/06-searching-sorting/readings/02-algorithm-analysis)
- Lecture Slides 6.03: Searching and Sorting with Alonzo (/bjc-course/curriculum/06-searching-sorting/readings/03-algorithms-slides.pdf)

External Resources

- How Algorithms Shape our World (http://www.ted.com/talks/kevin_slavin_how_algorithms_shape_our_world.html) - TED Talk

Labs/Exercises

- Activity 6.01: Searching Activity (/bjc-course/curriculum/06-searching-sorting/labs/01-searching-activity)
- Activity 6.02: Searching (non Video-) Game (/bjc-course/curriculum/06-searching-sorting/labs/02-searching-game)
- Lab 6.03: Update the Guessing Game (/bjc-course/curriculum/curriculum/06-searching-sorting/labs/03-update-guessing-game)

A FUTURIST PRECEDENT

In 1937, H. G. Wells anticipated Vannevar Bush's 1945 vision of a "memex." Wells wrote even more clearly about the possibility of indexing everything, and what that would mean for civilization:

There is no practical obstacle whatever now to the creation of an efficient index to all human knowledge, ideas and achievements, to the creation, that is, of a complete planetary memory for all mankind. And not simply an index; the direct reproduction of the thing itself can be summoned to any properly prepared spot. ... This in itself is a fact of tremendous significance. It foreshadows a real intellectual unification of our race. The whole human memory can be, and probably in a short time will be, made accessible to every individual. ... This is no remote dream, no fantasy.

Capabilities that were inconceivable then are commonplace now. Digital computers, vast storage, and high-speed networks make information search and retrieval necessary. They also make it possible. The Web is a realization of Bush's memex, and search is key to making it useful.

It Matters How It Works

How can Google or Yahoo! possibly take a question it may never have been asked before and, in a split second, deliver results from machines around the world? The search engine doesn't "search" the entire World Wide Web in response to your question. That couldn't possibly work quickly enough—it would take more than a tenth of a second just for bits to move around the earth at the speed of light. Instead, the search engine has *already* built up an index of web sites. The search engine does the best it can to find an answer to your query using its index, and then sends its answer right back to you.

To avoid suggesting that there is anything unique about Google or Yahoo!, let's name our generic search engine Jen. Jen integrates several different processes to create the illusion that you simply ask her a question and she gives back good answers. The first three steps have nothing to do with your particular query. They are going on repeatedly and all the time, whether anyone is posing any queries or not. In computer speak, these steps are happening in the *background*:

1. **Gather information.** Jen explores the Web, visiting many sites on a regular basis to learn what they contain. Jen revisits old pages because their contents may have changed, and they may contain links to new pages that have never been visited.
2. **Keep copies.** Jen retains copies of many of the web pages she visits. Jen actually has a duplicate copy of a large part of the Web stored on her computers.
3. **Build an index.** Jen constructs a huge index that shows, at a minimum, which words appear on which web pages.

When you make a query, Jen goes through four more steps, in the *foreground*:

4. **Understand the query.** English has lots of ambiguities. A query like “red sox pitchers” is fairly challenging if you haven’t grown up with baseball!
5. **Determine the relevance of each possible result to the query.** Does the web page contain information the query asks about?
6. **Determine the ranking of the relevant results.** Of all the relevant answers, which are the “best”?
7. **Present the results.** The results need not only to be “good”; they have to be shown to you in a form you find useful, and perhaps also in a form that serves some of Jen’s other purposes—selling more advertising, for example.

Each of these seven steps involves technical challenges that computer scientists love to solve. Jen’s financial backers hope that her engineers solve them better than the engineers of competing search engines.

We’ll go through each step in more detail, as it is important to understand what is going on—at every step, more than technology is involved. Each step also presents opportunities for Jen to use her information-gathering and editorial powers in ways you may not have expected—ways that shape your view of the world through the lens of Jen’s search results.

The background processing is like the set-building and rehearsals for a theatrical production. You couldn’t have a show without it, but none of it happens while the audience is watching, and it doesn’t even need to happen on any particular schedule.

Step 1: Gather Information

Search engines don't index everything. The ones we think of as general utilities, such as Google, Yahoo!, and Ask, find information rather indiscriminately throughout the Web. Other search engines are domain-specific. For example, Medline searches only through medical literature. ArtCylopedia indexes 2,600 art sites. The FindLaw LawCrawler searches only legal web sites. Right from the start, with any search engine, some things are in the index and some are out, because some sites are visited during the gathering step and others are not. Someone decides what is worth remembering and what isn't. If something is left out in Step 1, there is no possibility that you will see it in Step 7.

Speaking to the Association of National Advertisers in October 2005, Eric Schmidt, Google's CEO, observed that of the 5,000 terabytes of information in the world, only 170 terabytes had been indexed. (A *terabyte* is about a trillion bytes.) That's just a bit more than 3%, so 97% was not included. Another estimate puts the amount of indexed information at only .02% of the size of the databases and documents reachable via the Web. Even in the limited context of the World Wide Web, Jen needs to decide what to look at, and how frequently. These decisions implicitly define what is important and what is not, and will limit what Jen's users can find.

How *often* Jen visits web pages to index them is one of her precious trade secrets. She probably pays daily visits to news sites such as CNN.com, so that if you ask tonight about something that happened this morning, Jen may point you to CNN's story. In fact, there is most likely a master list of sites to be visited frequently, such as whitehouse.gov—sites that change regularly and are the object of much public interest. On the other hand, Jen probably has learned from her repeated visits that some sites don't change at all. For example, the Web version of a paper published ten years ago doesn't change. After a few visits, Jen may decide to revisit it once a year, just in case. Other pages may not be posted long enough to get indexed at all. If you post a futon for sale on Craigslist.com, the ad will become accessible to potential buyers in just a few minutes. If it sells quickly, however, Jen may never see it. Even if the ad stays up for a while, you probably won't be able to find it with most search engines for several days.

Jen is clever about how often she revisits pages—but her cleverness also codifies some judgments, some priorities—some *control*. The more important Jen judges your page to be, the less time it will take for your new content to show up as responses to queries to Jen's search engine.

Jen roams the Web to gather information by following links from the pages she visits. Software that crawls around the Web is (in typical geek

How A SPIDER EXPLORES THE WEB

Search engines gather information by wandering through the World Wide Web. For example, when a spider visits the main URL of the publisher of this book, www.pearson.com, it retrieves a page of text, of which this is a fragment:

```
<div id="subsidiary">
<h2 class="hide">Subsidiary sites links</h2>
<label for="subsidiarySites" class="hide">Available
sites</label>
<select name="subsidiarySites" id="subsidiarySites" size="1">
<option value="">Browse sites</option>
<optgroup label="FT Group">
<option value="http://www.ftchinese.com/sc/index.jsp">
    Chinese.FT.com</option>
<option value="http://ftd.de/">FT Deutschland</option>
```

This text is actually a computer program written in a special programming language called HTML ("HyperText Markup Language"). Your web browser renders the web page by executing this little program. But the spider is retrieving this text not to render it, but to index the information it contains. "FT Deutschland" is text that appears on the screen when the page is rendered; such terms should go into the index. The spider recognizes other links, such as www.ftchinese.com or ftd.de, as URLs of pages it needs to visit in turn. In the process of visiting those pages, it indexes them and identifies yet more links to visit, and so on!

A spider, or web crawler, is a particular kind of *bot*. A bot (as in "robot") is a program that endlessly performs some intrinsically repetitive task, often an information-gathering task.

irony) called a “spider.” Because the spidering process takes days or even weeks, Jen will not know immediately if a web page is taken down—she will find out only when her spider next visits the place where it used to be. At that point, she will remove it from her index, but in the meantime, she may respond to queries with links to pages that no longer exist. Click on such a link, and you will get a message such as “Page not found” or “Can’t find the server.”

Because the Web is unstructured, there is no inherently “correct” order in which to visit the pages, and no obvious way to know when to stop. Page A may contain references to page B, and also page B to page A, so the spider has to be careful not to go around in circles. Jen must organize her crawl of

the Web to visit as much as she chooses without wasting time revisiting sections she has already seen.

A web site may stipulate that it does not want spiders to visit it too frequently or to index certain kinds of information. The site's designer simply puts that information in a file named robots.txt, and virtually all web-crawling software will respect what it says. Of course, pages that are inaccessible without a login cannot be crawled at all. So, the results from Step 7 may be influenced by what the sites want Jen to know about them, as well as by what Jen thinks is worth knowing. For example, Sasha Berkovich was fortunate that the Polotsky family tree had been posted to part of the genealogy.com web site that was open to the public—otherwise, Google's spider could not have indexed it.

Finally, spidering is not cost free. Jen's "visits" are really requests to web sites that they send their pages back to her. Spidering creates Internet traffic and also imposes a load on the web server. This part of search engines' background processing, in other words, has unintended effects on the experience of the entire Internet. Spiders consume network bandwidth, and they may tie up servers, which are busy responding to spider requests while their ordinary users are trying to view their pages. Commercial search engines attempt to schedule their web crawling in ways that won't overload the servers they visit.

Step 2: Keep Copies

Jen downloads a copy of every web page her spider visits—this is what it means to "visit" a page. Instead of rendering the page on the screen as a web browser would, Jen indexes it. If she wishes, she can retain the copy after she has finished indexing it, storing it on her own disks. Such a copy is said to be "cached," after the French word for "hidden." Ordinarily Jen would not do anything with her cached copy; it may quickly become out of date. But caching web pages makes it possible for Jen to have a page that no longer exists at its original source, or a version of a page older than the current one. This is the flip side of Jen never knowing about certain pages because their owners took them down before she had a chance to index them. With a cached page, Jen knows what used to be on the page even after the owner intended it to disappear.

Caching is another blow to the Web-as-library metaphor, because removing information from the bookshelf doesn't necessarily get rid of it. Efforts to scrub even dangerous information are beyond the capability of those who posted it. For example, after 9/11, a lot of information that was once available on the Web was pulled. Among the pages that disappeared overnight

were reports on government vulnerabilities, sensitive security information, and even a Center for Disease Control chemical terrorism report that revealed industry shortcomings. Because the pages had been cached, however, the bits lived on at Google and other search engine companies.

Not only did those pages of dangerous information survive, but anyone could find them. Anytime you do a search with one of the major search engines, you are offered access to the cached copy, as well as the link to where the page came from, whether or not it still exists. Click on the link for the “Cached” page, and you see something that looks very much like what you might see if you clicked on the main link instead. The cached copy is identified as such (see Figure 4.3).

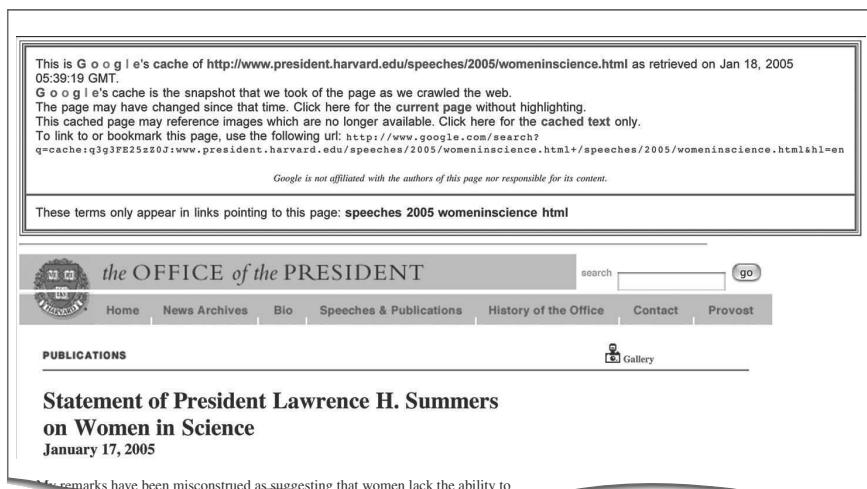


FIGURE 4.3 Part of a cached web page, Google's copy of an official statement made by Harvard's president and replaced two days later after negative public reaction. This copy was retrieved from Google after the statement disappeared from the university's web site. Harvard, which holds the copyright on this once-public statement, refused to allow it to be printed in this book (see Conclusion).

This is an actual example; it was the statement Lawrence Summers released on January 17, 2005, after word of his remarks about women in science became public. As reported in *Harvard Magazine* in March–April 2005, the statement began, “My remarks have been misconstrued as suggesting that women lack the ability to succeed at the highest levels of math and science. I did not say that, nor do I believe it.” This unapologetic denial stayed on the

The digital explosion grants the power of both instant communication and instant retraction—but almost every digital action leaves digital fingerprints.

carefully.” Those searching for the President’s statement were then led to the contrite new statement—but for a time, the original, defiant version remained visible to those who clicked on the link to Google’s cached copy.

FINDING DELETED PAGES

An easy experiment on finding deleted pages is to search using Google for an item that was sold on craigslist. You can use the “site” modifier in the Google search box to limit your search to the craigslist web site, by including a “modifier”:

futon site:craigslist.com

The results will likely return pages for items that are no longer available, but for which the cached pages will still exist.

what right does Jen have to show you her cached copy? For that matter, what right did she have to keep a copy in the first place? If you have copyrighted something, don’t you have some authority over who can make copies of it?

This enigma is an early introduction to the confused state of copyright law in the digital era, to which we return in Chapter 6. Jen cannot index my web page without receiving a copy of it. In the most literal sense, any time you “view” or “visit” a web page, you are actually copying it, and then your web browser renders the copy on the screen. A metaphorical failure once again: The Web is *not a library*. Viewing is an exchange of bits, not a passive activity, as far as the web site is concerned. If “copying” copyrighted materials was totally prohibited, neither search engines nor the Web itself could work, so some sort of copying must be permissible. On the other hand, when Jen caches the material she indexes—perhaps an entire book, in the case of the

Harvard web site for only a few days. In the face of a national firestorm of protest, Summers issued a new statement on January 19, 2005, reading, in part, “I deeply regret the impact of my comments and apologize for not having weighed them more

The digital explosion grants the power of both instant communication and instant retraction—but almost every digital action leaves digital fingerprints. Bits do not die easily, and digital words, once said, are hard to retract.

If Jen caches web pages, it may be possible for you to get information that was retracted after it was discovered to be in error or embarrassing. Something about this doesn’t feel quite right, though—is the information on those pages really Jen’s to do with as she wishes? If the material is copyrighted—a published paper from ten years ago, for example—

Google Books project—the legal controversies become more highly contested. Indeed, as we discuss in Chapter 6, the Association of American Publishers and Google are locked in a lawsuit over what Google is and is not allowed to do with the digital images of books that Google has scanned.

Step 3: Build an Index

When we searched the Web for “Zyprexa,” Jen consulted her index, which has the same basic structure as the index of a book: a list of terms followed by the places they occur. Just as a book’s index lists page numbers, Jen’s index lists URLs of web pages. To help the search engine give the most useful responses to queries, the index may record other information as well: the size of the font in which the term appears, for example, and where on the page it appears.

Indexes are critical because having the index in order—like the index of a book, which is in alphabetical order—makes it possible to find things much faster than with sequential searching. This is where Jen’s computer scientists really earn their salaries, by devising clever ways of storing indexed information so it can be retrieved quickly. Moore’s Law also played a big role in the creation of web indexes—until computer memories got fast enough, cheap enough, and big enough, even the cleverest computer scientists could not program machines to respond instantly to arbitrary English queries.

When Jen wants to find a term in her index, she does not start at the beginning and go through it one entry at a time until she finds what she is looking for. That is not the way you would look up something in the index of a book; you would use the fact that the index is in order alphabetically. A very simple strategy to look up something in a big ordered index, such as a phone book, is just to open the book in the middle and see if the item you are looking for belongs in the first half or the second. Then you can ignore half the phone book and use the same strategy to subdivide the remaining half. The number of steps it takes to get down to a single page in a phone book with n pages using this method is the number of times you have to divide n by 2 to get down to 1. So if n is 1000, it takes only 10 of these probing steps to find any item using *binary search*, as this method is known.

INDEXES AND CONCORDANCES

The information structure used by search engines is technically known as an *inverted index*—that is, an index of the words in a document or a set of documents, and the places where those words appear. Inverted indexes are not a new idea; the biblical concordances laboriously constructed by medieval monks were inverted indexes. Constructing concordances was one of the earliest applications of computer technology to a nonmathematical problem.

In general, the number of steps needed to search an index of n things using binary search is proportional, not to n , but to the number of digits in n . That means that binary search is exponentially faster than linear search—searching through a million items would take only 20 steps, and through a billion items would take 30 steps. And binary search is fairly dumb by comparison with what people actually do—if you were looking for “Leedeen” in the phone book, you might open it in the middle, but if you were looking for “Abelson,” you’d open it near the front. That strategy can be reduced to an even better computer algorithm, exponentially faster than binary search.

How big is Jen’s index, in fact? To begin with, how many terms does Jen index? That is another of her trade secrets. Jen’s index could be useful with a few tens of millions of entries. There are fewer than half a million words in the English language, but Jen probably wants to index some numbers too (try searching for a number such as 327 using your search engine). Proper names and at least some words in foreign languages are also important. The list of web pages associated with a term is probably on disk in most cases, with only the information about *where* on the disk kept with the term itself in main memory. Even if storing the term and the location on disk of the list of associated URLs takes 100 bytes per entry, with 25 million entries, the table of index entries would occupy 2.5 gigabytes (about 2.5 billion bytes) of main memory. A few years ago, that amount of memory was unimaginable; today, you get that on a laptop from Wal-Mart. The index can be searched quickly—using binary search, for example—although retrieving the list of URLs might require going to disk. If Jen has Google’s resources, she can speed up her query response by keeping URLs in main memory too, and she can split the search process across multiple computers to make it even faster.

Now that the preparations have been made, we can watch the performance itself—what happens when you give Jen a query.

Step 4: Understand the Query

When we asked Google the query *Yankees beat Red Sox*, only one of the top five results was about the Yankees beating the Red Sox (see Figure 4.4). The others reported instead on the Red Sox beating the Yankees. Because English is hard for computers to understand and is often ambiguous, the simplest form of query analysis ignores syntax, and treats the query as simply a list of keywords. Just looking up a series of words in an index is computationally easy, even if it often misses the intended meaning of the query.

To help users reduce the ambiguity of their keyword queries, search engines support “advanced queries” with more powerful features. Even the simplest, putting a phrase in quotes, is used by fewer than 10% of search

engine users. Typing the quotation marks in the query “Red Sox beat Yankees” produces more appropriate results. You can use “~” to tell Google to find synonyms, “-” to exclude certain terms, or cryptic commands such as “allinurl:” or “inanchor:” to limit the part of the Web to search. Arguably we didn’t ask our question the right way, but most of us don’t bother; in general, people just type in the words they want and take the answers they get.

Often they get back quite a lot. Ask Yahoo! for the words “allergy” and “treatment,” and you find more than 20,000,000 references. If you ask for “allergy treatment”—that is, if you just put quotes around the two words—you get 628,000 entries, and quite different top choices. If you ask for “treating allergies,” the list shrinks to 95,000. The difference between these queries may have been unintentional, but the search engine thought they were drastically different. It’s remarkable that human-computer communication through the lens of the search engine is so useful, given its obvious imperfections!

Web Images Maps News Shopping Gmail more ▾ Sign in

Google™ yankees beat red sox Search Advanced Search Preferences

Web Results 1 - 10 of about 220,000 for yankees beat red sox. (0.19 seconds)

ESPN - Boston's blow out caps unequalled comeback - MLB
MLB Recap: Believe it, New England, the **Red Sox** are in the World Series. ... We **beat** the **Yankees**. Now they get a chance to watch us on the tube." ...
sports.espn.go.com/mlb/recap?gameId=241020110 - 119k - [Cached](#) - [Similar pages](#)

ESPN - Yankees win eighth consecutive AL East title - MLB
The **Red Sox** recovered the next year, rallying from a 3-0 deficit in the AL championship series to celebrate on the field at **Yankee Stadium**, then sweeping ...
sports.espn.go.com/mlb/recap?gameId=251001102 - 118k - [Cached](#) - [Similar pages](#)

Red Sox rout Yankees, complete historic rally - Baseball- msnbc.com
The Boston **Red Sox** celebrate after defeating the New York **Yankees** 10-3 in Game 7 of ... We **beat** the **Yankees**. Now they get a chance to watch us on the tube." ...
www.msnbc.msn.com/id/6294431/ - 75k - [Cached](#) - [Similar pages](#)

Red Sox Beat Yankees! – Political Wire
Red Sox Beat Yankees! From the Boston Globe this morning: "The greatest comeback in sports history. Period. "Go ahead, find another one. It's impossible. ...
politicalwire.com/archives/2004/10/21/red_sox_beat_yankees.html - 18k - [Cached](#) - [Similar pages](#)

USATODAY.com - Red Sox stun Yankees to cap comeback, reach World ...
By defeating the **Yankees** in this year's ALCS, have the **Red Sox** put the 'Curse of the Bambino' ... Belhom: **Red Sox** were confident they'd **beat** the **Yankees** ...
www.usatoday.com/sports/baseball/playoffs/2004-10-20-redsox-yankees-game7_x.htm - 107k - [Cached](#) - [Similar pages](#)

Google™ is a registered trademark of Google, Inc. Reprinted by permission.

FIGURE 4.4 Keyword search misses the meaning of English-language query. Most of the results for the query “Yankees beat Red Sox” are about the Red Sox beating the Yankees.

NATURAL LANGUAGE QUERIES

Query-understanding technology is improving. The experimental site www.digger.com, for example, tells you when your query is ambiguous and helps you clarify what you are asking. If you ask Digger for information about "java," it realizes that you might mean the beverage, the island, or the programming language, and helps get the right interpretation if it guessed wrong the first time.

Powerset (www.powerset.com) uses natural language software to disambiguate queries based on their English syntax, and answers based on what web pages actually say. That would resolve the misunderstanding of "Yankees beat Red Sox."

Ongoing research promises to transfer the burden of disambiguating queries to the software, where it belongs, rather than forcing users to twist their brains around computerese. Natural language understanding seems to be on its way, but not in the immediate future. We may need a hundred-fold increase in computing power to make semantic analysis of web pages accurate enough so that search engines no longer give boneheaded answers to simple English queries.

Today, users tend to be tolerant when search engines misunderstand their meaning. They blame themselves and revise their queries to produce better results. This may be because we are still amazed that search engines work at all. In part, we may be tolerant of error because in web search, the cost to the user of an inappropriate answer is very low. As the technology improves, users will expect more, and will become less tolerant of wasting their time sorting through useless answers.

Step 5: Determine Relevance

A search engine's job is to provide results that match the intent of the query. In technical jargon, this criterion is called "relevance." Relevance has an objective component—a story about the Red Sox beating the Yankees is only marginally responsive to a query about the Yankees beating the Red Sox. But relevance is also inherently subjective. Only the person who posed the query can be the final judge of the relevance of the answers returned. In typing my query, I probably meant the New York Yankees beating the Boston Red Sox of Major League Baseball, but I didn't say that—maybe I meant the Flagstaff Yankees and the Continental Red Sox of Arizona Little League Baseball.

Finding all the relevant documents is referred to as “recall.” Because the World Wide Web is so vast, there is no reasonable way to determine if the search engine is finding everything that is relevant. Total recall is unachievable—but it is also unimportant. Jen could give us thousands or even millions more responses that she judges to be relevant, but we are unlikely to look beyond the first page or two. Degree of relevance always trumps level of recall. Users want to find a few good results, not all possible results.

The science of measuring relevance is much older than the Web; it goes back to work by Gerald Salton in the 1960s, first at Harvard and later at Cornell. The trick is to automate a task when what counts as success has such a large subjective component. We want the computer to scan the document, look at the query, do a few calculations, and come up with a number suggesting how relevant the document is to the query.

As a very simple example of how we might calculate the relevance of a document to a query, suppose there are 500,000 words in the English language. Construct two lists of 500,000 numbers: one for the document and one for the query. Each position in the lists corresponds to one of the 500,000 words—for example, position #3682 might be for the word “drugs.” For the document, each position contains a count of the number of times the corresponding word occurs in the document. Do the same thing for the query—unless it contains repeated words, each position will be 1 or 0. Multiply the lists for the document and the query, position by position, and add up the 500,000 results. If no word in the query appears in the document, you’ll get a result of 0; otherwise, you will get a result greater than 0. The more frequently words from the query appear in the document, the larger the results will be.

SEARCH ENGINES AND INFORMATION RETRIEVAL

Three articles offer interesting insights into how search engines and information retrieval work:

“The Anatomy of a Large-Scale Hypertextual Web Search Engine” by Sergey Brin and Larry Page was written in 2000 and gives a clear description of how the original Google worked, what the goal was, and how it was differentiated from earlier search engines.

“Modern Information Retrieval: A Brief Overview” by Amit Singhal was written in 2001 and surveys the IR scene. Singhal was a student of Gerry Salton and is now a Google Fellow.

“The Most Influential Paper Gerald Salton Never Wrote” by David Dubin presents an interesting look at some of the origins of the science.

Figure 4.5 shows how the relevance calculation might proceed for the query “Yankees beat Red Sox” and the visible part of the third document of Figure 4.4, which begins, “Red Sox rout Yankees” (The others probably contain more of the keywords later in the full document.) The positions in the two lists correspond to words in a dictionary in alphabetical order, from “ant” to “zebra.” The words “red” and “sox” appear two times each in the snippet of the story, and the word “Yankees” appears three times.

Lexicon:	ant, ..., beat, ..., defeating, ..., new, ..., patriots, ..., red, ..., sox, ..., Yankees, ..., zebra, ...
Doc:	0, ..., 1, ..., 2, ..., 1, ..., 0, ..., 2, ..., 2, ..., 3, ..., 0, ...
Query:	0, ..., 1, ..., 0, ..., 0, ..., 0, ..., 1, ..., 1, ..., 1, ..., 0, ...
Doc	
\times	0, ..., 1, ..., 0, ..., 0, ..., 0, ..., 2, ..., 2, ..., 3, ..., 0, ...
Query	
Sum of elements of Doc \times Query = $1+2+2+3 = 8$ = “relevance” of document to query	

FIGURE 4.5 Document and query lists for relevance calculation.

That is a very crude relevance calculation—problems with it are easy to spot. Long documents tend to be measured as more relevant than short documents, because they have more word repetitions. Uninteresting words such as “from” add as much to the relevance score as more significant terms such as “Yankees.” Web search engines such as Google, Yahoo!, MSN, and Ask.com consider many other factors in addition to which words occur and how often. In the list for the document, perhaps the entries are not word counts, but another number, adjusted so words in the title of the page get greater weight. Words in a larger font might also count more heavily. In a query, users tend to type more important terms first, so maybe the weights should depend on where words appear in the query.

Step 6: Determine Ranking

Once Jen has selected the relevant documents—perhaps she’s chosen all the documents whose relevance score is above a certain threshold—she “ranks” the search results (that is, puts them in order). Ranking is critical in making the search useful. A search may return thousands of relevant results, and users want to see only a few of them. The simplest ranking is by relevance—putting the page with the highest relevance score first. That doesn’t work well, however. For one thing, with short queries, many of the results will have approximately the same relevance.

More fundamentally, the documents Jen returns should be considered “good results” not just because they have high relevance to the query, but also because the documents themselves have high quality. Alas, it is hard to say what “quality” means in the search context, when the ultimate test of success is providing what people want. In the example of the earlier sidebar, who is to judge whether the many links to material about Britney Spears are really “better” answers to the “spears” query than the link to Professor Spears? And whatever “quality” may be, the ranking process for the major web search engines takes place automatically, without human intervention. There is no way to include protocols for checking professional licenses and past convictions for criminal fraud—not in the current state of the Web, at least.

Even though quality can’t be measured automatically, something like “importance” or “reputation” can be extracted from the structure of linkages that holds the Web together. To take a crude analogy, if you think of web pages as scientific publications, the reputations of scientists tend to rise if their work is widely cited in the work of other scientists. That’s far from a

WHAT MAKES A PAGE SEARCHABLE

No search provider discloses the full details of its relevance and ranking algorithm. The formulas remain secret because they offer competitive advantages, and because knowing what gives a page high rank makes abuse easier. But here are some of the factors that might be taken into account:

- Whether a keyword is used in the title of the web page, a major heading, or a second-level heading
- Whether it appears only in the body text, and if so, how “prominently”
- Whether the web site is considered “trustworthy”
- Whether the pages linked to from within the page are themselves relevant
- Whether the pages that link to this page are relevant
- Whether the page is old or young
- Whether the pages it links to are old or young
- Whether it passes some objective quality metric—for example, not containing any misspellings

Once you go to the trouble of crawling the Web, there is plenty to analyze, if you have the computing power to do it!

perfect system for judging the importance of scientific work—junk science journals do exist, and sometimes small groups of marginal scientists form mutual admiration societies. But for the Web, looking at the linkage structure is a place to start to measure the significance of pages.

One of Google's innovations was to enhance the relevance metric with another numerical value called “PageRank.” PageRank is a measure of the “importance” of each a page that takes into account the external references to it—a World Wide Web popularity contest. If more web pages link to a particular page, goes the logic, it must be more important. In fact, a page should be judged more important if a lot of *important* pages link to it than if the same number of unimportant pages link to it. That seems to create a circular definition of importance, but the circularity can be resolved—with a bit of mathematics and a lot of computing power.

This way of ranking the search results seems to reward reputation and to be devoid of judgment—it is a mechanized way of aggregating mutual opinions. For example, when we searched using Google for “schizophrenia drugs,” the top result was part of the site of a Swedish university. Relevance was certainly part of the reason that page came up first; the page was specifically about drugs used to treat schizophrenia, and the words “schizophrenia” and “drugs” both appeared in the title of the page. Our choice of words affected the relevance of the page—had we gone to the trouble to type “medicines” instead of “drugs,” this link wouldn’t even have made it to the first page of search results. Word order matters, too—Google returns different results for “drugs schizophrenia” than for “schizophrenia drugs.”

Sergey Brin and Larry Page, Google’s founders, were graduate students at Stanford when they developed the company’s early technologies. The “Page” in “PageRank” refers not to web pages, but to Larry Page.

This page may also have been ranked high because many other web pages contained references to it, particularly if many of those pages were themselves judged to be important. Other pages about schizophrenia drugs may have used better English prose style, may have been written by more respected scientific authorities, and may have contained more up-to-date

information and fewer factual errors. The ranking algorithm has no way to judge any of that, and no one at Google reads every page to make such judgments.

Google, and other search engines that rank pages automatically, use a secret recipe for ranking—a pinch of this and a dash of that. Like the formula

for Coca-Cola, only a few people know the details of commercial ranking algorithms. Google's algorithm is patented, so anyone can read a description. Figure 4.6 is an illustration from that patent, showing several pages with links to each other. This illustration suggests that both the documents themselves and the links between them might be assigned varying numbers as measures of their importance. But the description omits many details and, as actually implemented, has been adjusted countless times to improve its performance. A company's only real claim for the validity of its ranking formula is that people like the results it delivers—if they did not, they would shift to one of the competing search engines.

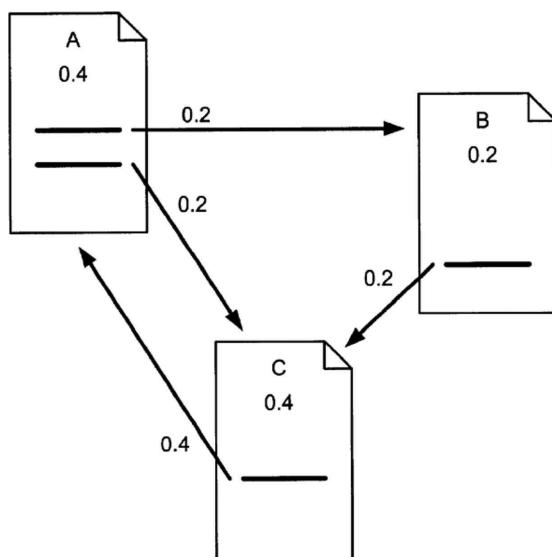


FIGURE 4.6 A figure from the PageRank patent (U.S. Patent #6285999), showing how links between documents might receive different weights.

It may be that one of the things people like about their favored search engine is consistently getting what they believe to be unbiased, useful, and even truthful information. But “telling the truth” in search results is ultimately only a means to an end—the end being greater profits for the search company.

Ranking is a matter of opinion. But a lot hangs on those opinions. For a user, it usually does not matter very much which answer comes up first or whether any result presented is even appropriate to the query. But for a

company offering a product, where it appears in the search engine results *can* be a matter of life and death.

KinderStart (www.kinderstart.com) runs a web site that includes a directory and search engine focused on products and services for young children. On March 19, 2005, visits to its site declined by 70% when Google lowered its PageRank to zero (on a scale of 0 to 10). Google may have deemed KinderStart's page to be low quality because its ranking algorithm found the page to consist mostly of links to other sites. Google's public description of its criteria warns about pages with "little or no original content." KinderStart saw matters differently and mounted a class action lawsuit against Google, claiming, among other things, that Google had violated its rights to free speech under the First Amendment by making its web site effectively invisible. Google countered that KinderStart's low PageRank was just Google's opinion, and opinions were not matters to be settled in court:

Google, like every other search engine operator, has made that determination for its users, exercising its judgment and expressing its opinion about the relative significance of web sites in a manner that has made it the search engine of choice for millions. Plaintiff KinderStart contends that the judiciary should have the final say over that editorial process.

SEEING A PAGE'S PAGERANK

Google has a toolbar you can add to certain browsers, so you can see PageRanks of web pages. It is downloadable from toolbar.google.com. You can also use the site www.iwebtool.com/pagerank_checker to enter a URL in a window and check its PageRank.

No fair, countered KinderStart to Google's claim to be just expressing an opinion. "PageRank," claimed KinderStart, "is not a mere statement of opinion of the innate value or human appeal of a given web site and its web pages," but instead is "a mathematically-generated product of measuring and assessing the quantity and depth of all the hyperlinks on the Web that tie into PageRanked web site, under programmatic determination by Defendant Google."

The judge rejected every one of KinderStart's contentions—and not just the claim that KinderStart had a free speech right to be more visible in Google searches. The judge also rejected claims that Google was a monopoly guilty of antitrust violations, and that KinderStart's PageRank of zero amounted to a defamatory statement about the company.

Whether it's a matter of opinion or manipulation, KinderStart is certainly much easier to find using Yahoo! than Google. Using Yahoo!, kinderstart.com is the top item returned when searching for "kinderstart." When we used Google, however, it did not appear until the twelfth page of results.

A similar fate befell bmw.de, the German web page of automaker BMW. The page Google indexed was straight text, containing the words "gebrauchtwagen" and "neuwagen" ("used car" and "new car") dozens of times. But a coding trick caused viewers instead to see a more conventional page with few words and many pictures. The effect was to raise BMW's position in searches for "new car" and "used car," but the means violated Google's clear instructions to web site designers: "Make pages for users, not for search engines. Don't deceive your users or present different content to search engines than you display to users, which is commonly referred to as 'cloaking.'" Google responded with a "death penalty"—removing bmw.de from its index. For a time, the page simply ceased to exist in Google's universe. The punitive measure showed that Google was prepared to act harshly against sites attempting to gain rank in ways it deemed consumers would not find helpful—and at the same time, it also made clear that Google was prepared to take *ad hoc* actions against individual sites.

Step 7: Presenting Results

After all the marvelous hard work of Steps 1–6, search engines typically provide the results in a format that is older than Aristotle—the simple, top-to-bottom list. There are less primitive ways of displaying the information.

If you search for something ambiguous like "washer" with a major web search engine, you will be presented with a million results, ranging from clothes washers to software packages that remove viruses. If you search Home Depot's web site for "washer," you will get a set of automatically generated choices to assist you in narrowing the search: a set of categories, price ranges, brand names, and more, complete with pictures (see Figure 4.7).

Alternatives to the simple rank-ordered list for presenting results better utilize the visual system. Introducing these new forms of navigation may shift the balance of power in the search equation. Being at the top of the list may no longer have the same economic value, but something else may replace the currently all-important rank of results—quality of the graphics, for example.

No matter how the results are presented, something else appears alongside them, and probably always will. It is time to talk about those words from the sponsors.

The screenshot shows the Home Depot website's search results for "washers". At the top, there's a navigation bar with links for Shopping Cart, Order Status, My List, My Registry, My Account, and Sign In. A "click here" button with a house icon and the text "GET INSPIRED" is also present. Below the navigation is a main menu with categories like Appliances, Bath, Building Supplies, Décor, Doors & Windows, Electronics, Flooring, Kitchen, Lighting & Fans, Outdoors, Paint, Storage, Tools & Hardware. A search bar with placeholder text "Enter Keyword or SKU" and a "SEARCH" button is located above the main content area. The breadcrumb trail indicates the user is at HOME > Text Search > washers.

Shop Online & Browse Store Products

Write a Review and You Can Win a \$100 Gift Card [Learn More](#)

Search Results

You Searched for "**washers**"

210 Results: 203 Products , 7 Articles

Matching Categories include:

- Appliances > Washers & Dryers
- Appliances > Washers & Dryers > Washers
- Building Supplies > Plumbing > Maintenance & Repair > Faucet > Washers
- Outdoors > Outdoor Power Equipment > Pressure Washers
- Outdoors > Outdoor Power Equipment > Pressure Washers > Pressure Washer Accessories

203 Products Sort By: Best Match

View Products in a: Grid | List Results per page: 12 ▾ 1 2 3 4 5 ▶

Select up to 4 items to compare. **▶ COMPARE**

| <input type="checkbox"/> Select to compare |
|---|---|---|---|
|  |  |  |  |
| Hot Washer Screw | Model TA-9 | Model WSSH300GWW | Model MTW600TQ |
| GE | GE 3.5 Cu. Ft. King-size Capacity Frontload Washer with Stainless Steel Basket | GE 3.2 Cu. Ft. Super Capacity Washer | Maytag® Maytag® Bravo High-Efficiency Top-Load Washer |
| \$3.44
Free Shipping | \$549.00 | \$319.00 | \$899.00 |

Source: Home Depot.

FIGURE 4.7 Results page from a search for “washers” on the Home Depot web site.

Who Pays, and for What?

Web search is one of the most widely used functions of computers. More than 90% of online adults use search engines, and more than 40% use them on a typical day. The popularity of search engines is not hard to explain. Search engines are generally free for anyone to use. There are no logins, no fine print to agree to, no connection speed parameters to set up, and no personal information to be supplied that you'd rather not give away. If you have an Internet connection, then you almost certainly have a web browser, and it probably comes with a web search engine on its startup screen. There are no directions

dominant advertising engine. Among the items and services for which Google will not accept advertisements are fake designer goods, child pornography (some adult material is permitted in the U.S., but not if the models *might* be underage), term paper writing services, illegal drugs and some legal herbal substances, drug paraphernalia, fireworks, online gambling, miracle cures, political attack ads (although political advertising is allowed in general), prostitution, traffic radar jammers, guns, and brass knuckles. The list paints a striking portrait of what Joe and Mary Ordinary want to see, should see, or will tolerate seeing—and perhaps also how Google prudentially restrains the use of its powerfully liberating product for illegal activities.

Search Is Power

At every step of the search process, individuals and institutions are working hard to control what we see and what we find—not to do us ill, but to help us. Helpful as search engines are, they don't have panels of neutral experts deciding what is true or false, or what is important or irrelevant. Instead, there are powerful economic and social motivations to present information that is to our liking. And because the inner workings of the search engines are not visible, those controlling what we see are themselves subject to few controls.

Algorithmic Does Not Mean Unbiased

Because search engines compute relevance and ranking, because they are “algorithmic” in their choices, we often assume that they, unlike human researchers, are immune to bias. But bias can be coded into a computer program, introduced by small changes in the weights of the various factors that go into the ranking recipe or the spidering selection algorithm. And even what *counts* as bias is a matter of human judgment.

Having a lot of money will not buy you a high rank by paying that money to Google. Google's PageRank algorithm nonetheless incorporates something of a bias in favor of the already rich and powerful. If your business has become successful, a lot of other web pages are likely to point to yours, and that increases your PageRank. This makes sense and tends to produce the results that most people feel are correct. But the degree to which power should beget more power is a matter over which powerful and marginal businesses might have different views. Whether the results “seem right,” or the search algorithm's parameters need adjusting, is a matter only humans can judge.

For a time, Amazon customers searching for books about abortion would get back results including the question, “Did you mean adoption?” When a pro-choice group complained, Amazon responded that the suggestion was automatically generated, a consequence of the similarity of the words. The search engine had noticed, over time, that many people who searched for “abortion” also searched for “adoption.” But Amazon agreed to make the *ad hoc* change to its search algorithm to treat the term “abortion” as a special case. In so doing, the company unintentionally confirmed that its algorithms sometimes incorporate elements of human bias.

Market forces are likely to drive commercially viable search engines toward the bias of the majority, and also to respond to minority interests only in proportion to their political power. Search engines are likely to favor fresh items over older and perhaps more comprehensive sources, because their users go to the Internet to get the latest information. If you rely on a search engine to discover information, you need to remember that others are making judgment calls for you about what you are being shown.

Not All Search Engines Are Equal

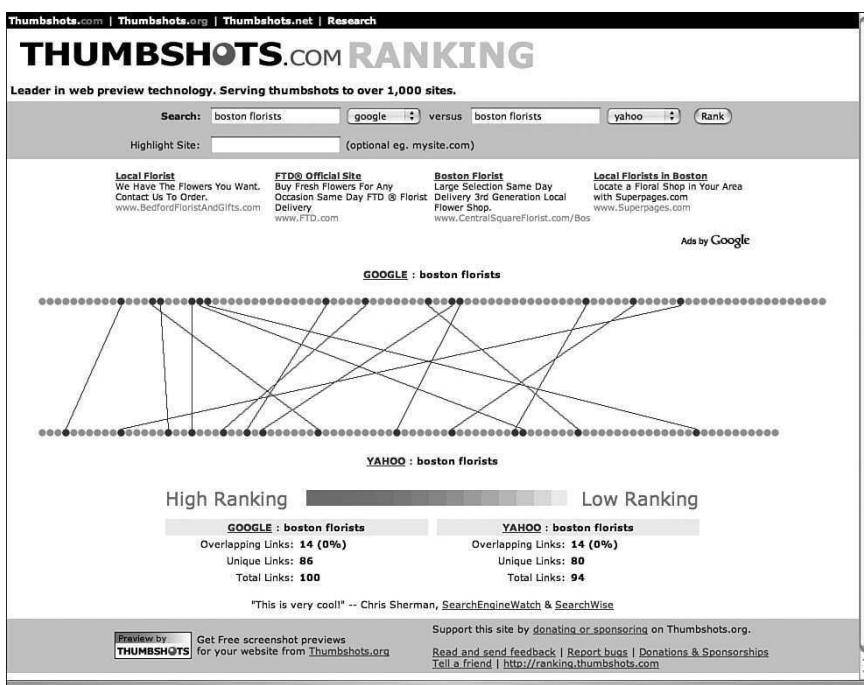
When we use a search engine, we may think that what we are getting is a representative sample of what’s available. If so, what we get from one search engine should be pretty close to what we get from another. This is very far from reality.

A study comparing queries to Google, Yahoo!, ASK, and MSN showed that the results returned on the first page were unique 88% percent of the time. Only 12% of the first-page results were in common to even two of these four search engines. If you stick with one search engine, you could be missing what you’re looking for. The tool ranking.thumbshots.com provides vivid graphic representations of the level of overlap between the results of different search engines, or different searches using the same search engine. For example, Figure 4.9 shows how little overlap exists between Google and Yahoo! search results for “boston florist.”

Each of the hundred dots in the top row represents a result of the Google search, with the highest-ranked result at the left. The bottom row represents Yahoo!’s results. A line connects each pair of identical search results—in this case, only 11% of the results were in common. Boston Rose Florist, which is Yahoo’s number-one response, doesn’t turn up in Google’s search at all—not in the top 100, or even in the first 30 pages Google returns.

Ranking determines visibility. An industry research study found that 62% of search users click on a result from the first page, and 90% click on a result within the first three pages. If they don’t find what they are looking for, more

than 80% start the search over with the same search engine, changing the keywords—as though confident that the search engine “knows” the right answer, but they haven’t asked the right question. A study of queries to the Excite search engine found that more than 90% of queries were resolved in the first three pages. Google’s experience is even more concentrated on the first page.



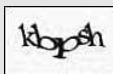
Reprinted with permission of SmartDevil, Inc.

FIGURE 4.9 Thumbshots comparison of Google and Yahoo! search results for “boston florists.”

Search engine users have great confidence that they are being given results that are not only useful but authoritative. 36% of users thought seeing a company listed among the top search results indicated that it was a top company in its field; only 25% said that seeing a company ranked high in search results would not lead them to think that it was a leader in its field. There is, in general, no reason for such confidence that search ranking corresponds to corporate quality.

CAT AND MOUSE WITH BLOG SPAMMERS

You may see comments on a blog consisting of nothing but random words and a URL. A malicious bot is posting these messages in the hope that Google's spider will index the blog page, including the spam URL. With more pages linking to the URL, perhaps its PageRank will increase and it will turn up in searches. Blogs counter by forcing you to type some distorted letters—a so-called *captcha* ("Completely Automated Public Turing test to tell Computers and Humans Apart"), a test to determine if the party posting the comment is really a person and not a bot. Spammers counter by having their bot take a copy of the captcha and show it to human volunteers. The spam bot then takes what the volunteers type and uses it to gain entry to the blog site. The volunteers are recruited by being given access to free pornography if they type the captcha's text correctly! Here is a sample captcha:



This image has been released into the public domain by its author, Kruglov at the wikipedia project. This applies worldwide.

Search Results Can Be Manipulated

Search is a remarkable business. Internet users put a lot of confidence in the results they get back from commercial search engines. Buyers tend to click on the first link, or at least a link on the first page, even though those links may depend heavily on the search engine they happen to be using, based on complex technical details that hardly anyone understands. For many students, for example, the library is an information source of last resort, if that. They do research as though whatever their search engine turns up must be a link to the truth. If people don't get helpful answers, they tend to blame themselves and change the question, rather than try a different search engine—even though the answers they get can be inexplicable and capricious, as anyone googling "kinderstart" to find kinderstart.com will discover.

Under these circumstances, anyone putting up a web site to get a message out to the world would draw an obvious conclusion. Coming out near the top of the search list is too important to leave to chance. Because ranking is algorithmic, a set of rules followed with diligence and precision, it must be possible to manipulate the results. The Search Engine Optimization industry (SEO) is based on that demand.

Search Engine Optimization is an activity that seeks to improve how particular web pages rank within major search engines, with the intent of

increasing the traffic that will come to those web sites. Legitimate businesses try to optimize their sites so they will rank higher than their competitors. Pranksters and pornographers try to optimize their sites, too, by fooling the search engine algorithms into including them as legitimate results, even though their trappings of legitimacy are mere disguises. The search engine companies tweak their algorithms in order to see through the disguises, but their tweaks sometimes have unintended effects on legitimate businesses. And the tweaking is largely done in secret, to avoid giving the manipulators any ideas about countermeasures. The result is a chaotic battle, with innocent bystanders, who have become reliant on high search engine rankings, sometimes injured as the rules of engagement keep changing.

Google proclaims of its PageRank algorithm that “Democracy on the web works,” comparing the ranking-by-inbound-links to a public election. But the analogy is limited—there are many ways to manipulate the “election,” and the voting rules are not fully disclosed.

The key to search engine optimization is to understand how particular engines do their ranking—what factors are considered, and what weights they are given—and then to change your web site to improve your score. For example, if a search engine gives greater weight to key words that appear in the title, and you want your web page to rank more highly when someone searches for “cameras,” you should put the word “cameras” in the title. The weighting factors may be complex and depend on factors external to your own web page—for example, external links that point to your page, the age of the link, or the prestige of the site from which it is linked. So significant time, effort, and cost must be expended in order to have a meaningful impact on results.

Then there are techniques that are sneaky at best—and “dirty tricks” at worst. Suppose, for example, that you are the web site designer for Abelson’s, a new store that wants to compete with Bloomingdale’s. How would you entice people to visit Abelson’s site when they would ordinarily go to Bloomingdale’s? If you put “We’re better than Bloomingdale’s!” on your web page, Abelson’s page might appear in the search results for “Bloomingdale’s.” But you might not be willing to pay the price of mentioning the competition on Abelson’s page. On the other hand, if you just put the word “Bloomingdale’s” *in white text on a white background* on Abelson’s page, a human viewer wouldn’t see it—but the indexing software might index it anyway. The indexer is working with the HTML code that generates the page, not the visible page itself. The software might not be clever enough to realize that the word “Bloomingdale’s” in the HTML code for Abelson’s web page would not actually appear on the screen.

A huge industry has developed around SEO, rather like the business that has arisen around getting high school students packaged for application to college. A Google search for “search engine optimization” returned 11 sponsored links, including some with ads reading “Page 1 Rankings Guarantee” and “Get Top Rankings Today.”

Is the search world more ethical because the commercial rank-improving transactions are indirect, hidden from the public, and going to the optimization firms rather than to the search firms? After all, it is only logical that if you have an important message to get out, you would optimize your site to do so. And you probably wouldn’t have a web site at all if you thought you had nothing important to say. Search engine companies tend to advise their web site designers just to create better, more substantive web pages, in much the same way that college admissions officials urge high school students just to learn more in school. Neither of the dependent third-party “optimization” industries is likely to disappear anytime soon because of such principled advice.

And what’s “best”—for society in general, not just for the profits of the search companies or the companies that rely on them—can be very hard to say. In his book, *Ambient Findability*, Peter Morville describes the impact of search engine optimization on the National Cancer Institute’s site, www.cancer.gov. The goal of the National Cancer Institute is to provide the most reliable and the highest-quality information to people who need it the most,

GOOGLE BOMBING

A “Google bomb” is a prank that causes a particular search to return mischievous results, often with political content. For example, if you searched for “miserable failure” after the 2000 U.S. presidential election, you got taken to the White House biography of George Bush. The libertarian Liberty Round Table mounted an effort against the Center for Science in the Public Interest, among others. In early 2008, www.libertyroundtable.org read, “Have you joined the Google-bombing fun yet? Lob your volleys at the food nazis and organized crime. Your participation can really make the difference with this one—read on and join the fun! Current Target: Verizon Communications, for civil rights violations.” The site explains what HTML code to include in your web page, supposedly to trick Google’s algorithms.

Marek W., a 23-year-old programmer from Cieszyn, Poland, “Google bombed” the country’s president, Lech Kaczyński. Searches for “kutas” using Google (it’s the Polish word for “penis”) returned the president’s web site as the first choice. Mr. Kaczyński was not pleased, and insulting the president is a crime in Poland. Marek is now facing three years in prison.

often cancer sufferers and their families. Search for “cancer,” and the NCI site was “findable” because it appeared near the topic of the search page results. That wasn’t the case, though, when you looked for specific cancers, yet that’s exactly what the majority of the intended users did. NCI called in search engine optimization experts, and all that is now changed. If we search for “colon cancer,” the specific page on the NCI site about this particular form of cancer appears among the top search results.

Is this good? Perhaps—if you can’t trust the National Cancer Institute, who *can* you trust? But WebMD and other commercial sites fighting for the top position might not agree. And a legitimate coalition, the National Colorectal Cancer Roundtable, doesn’t appear until page 7, too deep to be noticed by almost any user.

Optimization is a constant game of cat and mouse. The optimizers look for better ways to optimize, and the search engine folks look for ways to produce more reliable results. The game occasionally claims collateral victims. Neil Montcrief, an online seller of large-sized shoes, prospered for a while because searches for “big feet” brought his store, 2bigfeet.com, to the top of the list. One day, Google tweaked its algorithm to combat manipulation. Montcrief’s innocent site fell to the twenty-fifth page, with disastrous consequences for his economically marginal and totally web-dependent business.

Manipulating the ranking of search results is one battleground where the power struggle is played out. Because search is the portal to web-based information, controlling the search results allows you, perhaps, to control what people think. So even governments get involved.

Search Engines Don’t See Everything

Standard search engines fail to index a great deal of information that is accessible via the Web. Spiders may not penetrate into databases, read the contents of PDF or other document formats, or search useful sites that require a simple, free registration. With a little more effort than just typing into the search window of Google or Yahoo!, you may be able to find exactly what you are looking for. It is a serious failure to assume that something is unimportant or nonexistent simply because a search engine does not return it. A good overview of resources for finding things in the “deep web” is at Robert Lackie’s web site, www.robertlackie.com.

Search Control and Mind Control

To make a book disappear from a library, you don’t have to remove it from the bookshelf. All you need to do is to remove its entry from the library

All algorithms are not equal

This activity introduces the concept that algorithms are used to develop and express solutions to computational problems. It focus, in part, on the following learning objectives:

- 15: The student can develop an algorithm.
- 16: The student can express an algorithm in a language.
- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Comparing Algorithms

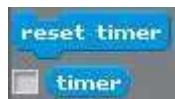
We all know that computers can be really fast at solving problems: feed a computer a problem that adds several million-digit numbers, and the answer comes out faster than a human has finished the first ten digits. (Well, unless you're one of those mental calculators (http://en.wikipedia.org/wiki/Mental_calculator).) In fact, that is one of the main motivations behind building computers: there are problems that humans can solve for small values, but once the values become large, it is better to ask a computer to do them instead, since a computer can crunch through calculations faster. Nonetheless, for many applications, speed is essential, and we can't trust a human to perform the calculations: GPS (http://en.wikipedia.org/wiki/Global_Positioning_System), for example, would be a very different (maybe non-existent) system if a human were continuously determining the position of a plane or of a car.

In these applications, the faster, the better, so if we're faced with two algorithms to perform the same problem, we want to be able to compare which one performs better. Since computers are more useful for large amounts of data, we also want to compare the performances for large inputs. One obvious way to do this is to run both algorithms on large inputs, time each of the algorithms from start to finish, and see which one performs better. (We can also see which algorithm takes up more space in memory, but for today's lab, we will only focus on time.)

Can you think of any other tasks where speed is essential? In other words, can you think of any task that probably would not exist (or that would be very tedious!) if a human were in charge of performing it?

Time is of the Es-sense

We have seen how BYOB can be used to wait for a certain time before reporting anything. BYOB also allows us to do the converse: to report how long a program takes to finish. In the `Sensing` menu, you will see a command called `reset timer` and a reporter called `timer`:

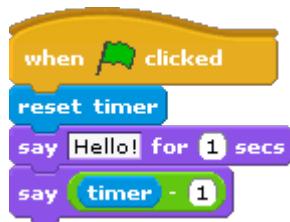


Activate (tick-mark) the timer reporter and you should see a timer ticking away above Alonzo. It's been ticking ever since you opened up BYOB, and counts in tenths of a second:

timer 134.0



We are going to be performing timing experiments in the following sections, so this timer will prove useful. Add the following timing framework script into the space for BYOB scripts:



The script will reset the timer whenever the green flag is pressed. Alonso will then say Hello! for 1 second, and soon after that, he will say the current time, less 1 second to account for how long he said Hello!. If we replace Hello! with a reporter, then Alonso will say the answer of the reporter, and one second later, how long the reporter took to generate the answer: this is the timing information that we need. Save the script with the name `TimingFramework`.

Do You Have Time to Add?

Let us start our timing experiments with something simple: how long does it take for a computer to add 1 to (or *increment*) a number? Replace Hello! in the timing script with an addition operator from the `Operators` menu, and add 1 to 100000 (that's five zeros!). Run the script a few times (around four) to get an idea of the average approximate time (in seconds) it takes for the computer to increment 100000. Run the script repeatedly by double-clicking on it; do not place the script inside a repeat or a repeat until block, since we are interested in knowing the approximate time the script takes to run once.

Now, pause here and think: what's your gut feeling for how much longer the computer would take if we doubled 100000? Test your intuition: Get an average approximate time for how long it takes the computer to increment numbers that you progressively double: 200000, 400000, 800000, and 1600000. Remember that for each number, you need to run the timing script multiple times (around four) to get an idea of the average approximate time (you don't have to be precise). What do you observe?

Take another huge leap and find out how long it (approximately) takes for the computer to increment numbers that you progressively scale by 10: 160000000, 1600000000 (that's eight zeros), and maybe 16000000000 (that's nine zeros). What do you observe?

Constant-Time

You may have noticed that the computer takes approximately the same time to increment a number, even though the number was made progressively larger. Computer scientists (programmers and theorists) thus call incrementing a number a **constant-time** operation. In fact, any basic arithmetic operation (addition, subtraction, multiplication, division, and exponentiation) is considered to be a constant-time operation.

Notice that we call these operations *constant-time* operations, but we don't actually say how much time they take. Different computers will take different amounts of time to perform the same operation. To report the exact time of any algorithm, we would have to also report the physical configurations of the computer that the algorithm was run on. This gets very difficult and annoying very fast, especially with the almost infinite variety of computers available today. Instead, we focus on how the running time of an algorithm scales as we scale its inputs to larger and larger sizes, because this is a property of the algorithm itself, and is independent of the computer that it is run on.

Why did the computer take approximately the same amount of time to increment a number, even though that number was getting larger? Think about how you would add one to a number, back from your elementary school days; this is similar to how a computer does its arithmetic (ignoring technical details). The elementary school way of adding numbers goes digit by digit, and so the amount of time it takes for you to add two numbers depends on how many digits each number has. As we doubled the number we were incrementing, we didn't consistently add digits to it, and so the computer took approximately the same time. Even as we began scaling the number by ten, the computer (and you!) takes a relatively really small amount of time to account for the extra digit, so the total time remains approximately constant.

Constant-time operations are the Holy Grail of computer science algorithms, and unfortunately, most algorithms are not constant-time, as we will soon see.

All the Numbers, All the Time

For this section, we will be using this timing framework (`./code/timing.ypr`). It is very similar to the framework you constructed earlier, but it also has the stubs for a few blocks that we will be filling in. The first block that we will fill in is already on stage:

insert all numbers between 1 and 10 into 

This block should fill the input list with all of the numbers from start to end. If working correctly, the version used on stage should generate all of the numbers from 1 to max, where max is a number that we will change when running timing experiments, and should add them to the numbers list.

First, talk to your partner and discuss an algorithm that you can use to fill the input list with all of the numbers from start to end. Once you both have decided on an algorithm, complete the body of the block. Now, run the script with max set to 10. Run it a few times to get an idea of the average time the computer takes to generate this list (to the nearest tenth of a second); you don't need to be exact! Repeat the experiment with max set to 20, 40, 100, 200, and 1000. Do you see a pattern?

Linear Time

You may have noticed that if you increased max by a factor of two, then the computer took approximately twice as much time to fill the numbers list. Similarly, if you increased max by a factor of ten, then the computer took approximately ten times as much time; if you increased max by a factor of 100, then the computer ran approximately 100 times longer. In general, as we scale the size of the input by a certain amount, we also scale the running time by the same amount. We call these algorithms **linear-time** algorithms, because if we were to plot the runtime of one such algorithm against the size of its input, we would get a line.

Why is the algorithm a linear-time algorithm? When max was set to 20, we had to add 20 numbers to the numberslist; when max was set to 40, we had to add 40 numbers to the number list. In other words, as we scaled max, we also scaled how many numbers we had to add to the numbers list by the same amount. Linear-time algorithms are also much sought-after in computer science, and for many problems, they are the fastest algorithms you can find.

Algorithm Analysis

This lecture focuses on algorithm analysis, focusing on the following learning objectives:

- 17: The student can appropriately connect problems and potential algorithmic solutions.
- 18: The student can evaluate algorithms analytically and empirically.

Introduction

An algorithm's **correctness** refers to whether or not it contains errors. We will talk about testing procedures below.

An algorithm's **clarity** refers to how well it is expressed and described. Can you understand what it does and how it does it? We will begin using **comments** to document our code.

An algorithm's **efficiency** refers generally to how efficiently it uses two key resources, time and space – i.e., the *processor or central processing unit* (CPU), and the computer's *main memory*, often called *random access memory* (RAM).

The faster the algorithm – the quicker it finishes its task – the more efficient it is with respect to time.

The less memory it uses, the more efficient it is with respect to space.

Sorting and Searching Algorithms

In discussing this topic we will talk about *sorting* and *searching* algorithms. A **sort** algorithm arranges data in some order – e.g., numeric or alphabetic order. A **search** algorithm looks up some target data – e.g., a name in the phone book. You can think of the data as being contained in a **list**.

Efficiency and Input Size

Efficiency issues only come into play when there are large amounts of **data**. An algorithm's *input size* is often abbreviated with *N*. As *N* grows, the differences between efficient and inefficient algorithms become clear.

For example, this table shows the differences in running time in seconds for the same sort algorithm on different input sizes.

List Size, N	Older Computer	Newer Computer
125	12.5	2.8
250	49.3	11.0
500	195.8	43.4
1000	780.3	172.9
2000	3114.9	690.5

Computer scientists like to *abstract away* the efficiency differences that are caused by different hardware and software – e.g., faster vs. slower computer.

Here are some old data obtained on different machines when sorting 2000 integers using the same algorithm:

Type of computer	Time
Home PC	51.915

Type of computer	Time
Desktop PC	11.508
Minicomputer	2.382
Mainframe	0.431
Supercomputer	0.087

Empirical Analysis: The data in both of these tables were gathered **empirically** or **experimentally** – i.e., by running and timing the actual program on different machines and inputs. Abstract Analysis

We can also analyze algorithm efficiency more abstractly, by comparing an algorithm to well known mathematical functions.

Consider the following graph with four functions : the logarithmic, linear, quadratic, and exponential functions:

- Logarithm function: $y = \log_2 x$
- Linear function: $y = 5x$
- Quadratic function: $y = x^2$
- Exponential function: $y = 2^x$

Graph

As you can see, as X increases, the running time increases. But look at the differences. Look at how slowly running time increases for the logarithmic function compared to the exponential function.

In terms of these four types of functions, we arrange them in the following order in terms of how fast they increase as the size of, x , increases:

logarithmic	linear	quadratic	exponential
$\log_2 x$	x	x^2	2^x

Algorithm Efficiency

Computer scientists like to characterize algorithm efficiency in terms of **how fast the running time will increase as the size of the inputs, n , increases**.

To see the dramatic differences, consider some numbers for these four types of functions:

Inputs	Logarithmic	Linear	Quadratic	Exponential
n	\log_2	n	n^2	2^n
8	3	8	64	256
16	4	16	256	65,536
32	5	32	1,024	4,296,967,296
64	6	64	4,096	1.84×10^{19}
128	7	128	16,384	3.40×10^{38}
256	8	256	65,536	1.15×10^{77}
512	9	512	262,144	1.34×10^{154}

So, for 512 inputs, a linear algorithm could perform some task in 512 seconds. Whereas an exponential algorithm would take 1.34×10^{154} seconds.

Algorithms and Problems

Computer scientists like to analyze problems by asking, **what type of algorithm do we need to solve this problem?** Or, what's the fastest type of algorithm that can solve this problem?

A Linear Search Algorithm

For example, to find the number 100 in the following lists of numbers, we would have to go through each number in the list, from left to right.

```
List 1 (16 numbers): (15 20 4 2 1 17 19 25 100 65 78 19 20 15 0 72)
```

```
List 2 (16 numbers): (15 20 4 2 1 17 19 25 65 78 19 20 15 0 72 75)
```

We can use a linear search algorithm:

```
# Search for X in List L and return its Index, indx, in L  
# Or return -1 if X is not in the List  
  
To Search for X in List L, DO:  
    Set global indx to 1  
    For each number, num, in the list, L:  
        If num = X:  
            Return indx      Set indx to indx + 1  
        Return -1          # The number is not in the list so return -1
```

TODO: Hand trace this algorithm on List 1 and List 2 and count how many times you have to go through the loop.

QUESTION: Suppose there 512 numbers in our list. In the **worst case** – i.e., when X is not in the list – how many times would we go through the while loop?

This algorithm (and the problem it solves) are said to be **linear** because in the **worst case** it takes N iterations of the loop to find something in a list of N items.

A Logarithmic Search Algorithm

If our list were sorted – e.g., like the telephone book – then it wouldn't make sense to do a linear search – i.e., start at page 1.

```
List 1 (16 numbers): (1 2 5 6 9 12 15 16 32 64 100 128 256 299 512 568)
```

```
List 2 (16 numbers): (1 2 5 6 9 12 15 16 32 64 99 128 256 299 512 568)
```

Instead we could guess the approximate location of the number in the list and look there. In the most general case, we could guess that the X was the middle value in the list. If X was greater (or smaller) than the middle value, we would search the top (bottom) half of the list.

This is known as the binary search algorithm. For example, to guess a secret number between 1 and N:

```
Guess the middle value in the range 1..N  
If the guess is too high,  
    cut off the top of the range.  
If the guess is too low,  
    cut off the bottom of the range.  
Repeat until there's only 1 number left.
```

QUESTION: How many guesses to find the secret number between 1 and 100? Between 1 and 500?

You should be able to see that as N grows larger, the binary search will grow very slowly, like a logarithmic function.

A Quadratic Sort Algorithm

Let's look at an example of an algorithm that performs like a quadratic function. Sort a list of N numbers from low to high using bubble sort.

List 1 (5 numbers): (10 5 4 4 1)

NOTE: Bubble sort is probably the world's worst sorting algorithm. And there are many sorting algorithms that are much more efficient. But here is bubble sort:

```
# Sort the N numbers in List L into ascending order

For a list of N items, repeat N-1 times.      #  N-1 steps
  For each adjacent pair of items           #  N-1 pairs
    If the pair is out of order, swap the numbers
```

This algorithm contains a **nested loop**. The outer loop repeats N-1 times. And on each repetition the inner loop does N-1 comparisons of adjacent numbers. So that is $N^2 - 2N + 1$ comparisons. That's quadratic.

Clock analogy: Think of the relationship between the second hand and the minute hand on a clock. The second hand ticks 60 times each minute. To time 1 hour, the minute hand would tick 60 times, which is 3600 seconds.

TODO: Hand trace this algorithm on List 1 and count how many times you have perform the IF statement?

Question: Suppose there are 512 numbers in the list and it took 1 second to compare and swap a pair of numbers. Approximately how many seconds would it take to sort the list?

Efficient Sorting

Bubble sort is an example of an in-place sorting algorithm. It doesn't require much more memory than the list itself. The most efficient in-place sorting algorithms can sort N numbers in time that is proportional to an $n \log n$ function. For 512 numbers that would require time proportional to $512 \times 8 = 8,192$ seconds as opposed to $512 \times 512 = 262,144$ seconds.

A Linear Sorting Algorithm

What if we didn't care about conserving space? How fast could we sort N numbers?

```
# Sort the list of N numbers in List L in ascending order. Suppose
# the list contained numbers whose values ranged between 1 and M.
# For example the numbers might be between 1 and 1000.

(1) Make a list, LL, with M "buckets", each containg an empty list.
(2) For each of the N numbers in list L, put it in its proper bucket.
(3) Traverse through the list, LL, and empty the buckets back into the list, L.
```

For example, for our original list, (10 5 4 4 1) and assuming that the numbers range between 1 and 10:

```
(0) L = ( 10 5 4 4 1 )
(1) LL = ( () () () () () () () () () )
(2) LL = ( (1) () () (4 4) (5) () () () (10) )
(3) L = ( 1 4 4 5 10 )
```

Questions

- **Space efficiency:** In terms of space, this algorithm requires space for the N numbers in L and the M buckets, so $N + M$. How much more space is this? Linearly more? Quadratically more?
- **Time efficiency:** In terms of our N inputs, this algorithm grows linearly as the size of N grows.

This shows that for a given problem, different algorithms can have different efficiencies.

An Exponential Algorithm

The Traveling Salesman Problem (<http://turing.cs.trincoll.edu/~ram/cpsc110/inclass/alg-analysis/>) is an example of a problem that can only be solved using an *exponential algorithm*. Given N cities, find the shortest path (least costly path) through all N cities.

The only known way to solve this problem is by **brute force**. List all possible paths through the cities. For 3 cities we can easily do this:

```
A B C
A C B
B A C
B C A
C B A
C A B
```

In general there are $N!$ ways to arrange N cities. So there are $3 \times 2 \times 1 = 6$ ways to arrange 3 cities. And in general $N!$ is much greater than $2N$:

N	$N!$	2^N
3	6	8
4	24	16
5	120	32
6	720	64

Problems that can only be solved by *exponential algorithms* are known as **intractable problems**. There is no general, optimal solution for the traveling salesman problem that runs in a reasonable amount of time.

There are **heuristic** solutions. A *heuristic* is a rule-of-thumb that gives a pretty good (but not always optimal) solution. For the traveling salesman problem we can use the nearest-neighbor heuristic – i.e., select the nearest neighboring city as the next city. Computability

Theoretically speaking, not practically, are there problems that cannot, in principle, be computed? We're not talking here about *practical* limits to computation.

The answer is “No”. In 1936 British Mathematician Alan Turing (http://en.wikipedia.org/wiki/Alan_Turing) proved that there is a set of problems that cannot be solved by any algorithm or computational procedure.

Example: The Halting Problem (http://en.wikipedia.org/wiki/Halting_problem). Given a description of a computer

program, determine whether the program halts or continues running forever. In other words, given a program and an set of inputs, decide whether the program will eventually halt given that set of inputs.

Turing proved that there is no general algorithm that can solve the halting problem for any program and any inputs.

We won't go into the details here, but it's important that you know that computers do have this limitation.

Exercises

Do the following exercises and add your answers to your Portfolio page for today's assignment.

For each of the following problems, decide whether it can be solved by a *linear* algorithm and explain briefly why or why not.

- Compute the sum of a list containing N random integers.
- Determining if duplicate numbers occur in a list of N random integers.

For each of the following problems, decide whether it can be solved by a *logarithmic algorithm* and explain briefly why or why not.

- Guessing a number between 1 and 100 if your guesses are characterized as "too high" or "too low".
- Guessing a number between 1 and 100 if your guesses are characterized as "wrong" or "right".

For each of the following problems, decide whether or not it is *intractable* and explain briefly why or why not.

- Computing the sum of all the numbers in a square matrix of dimension $N \times N$.
 - Given n integers does there exist a subset of them that sum exactly to B? For example, suppose the integers are {4, 5, 8, 13, 15, 24, 33}. If B = 36 then the answer is yes (and 4, 8, 24 is a solution). If B = 14 the answer is no.
-

SEARCHING & SORTING

Computer Science Principles

LEARNING OBJECTIVES

- 1: The student can use computing tools and techniques to create artifacts
- 3: The student can use computing tools and techniques for creative expression.
- 16: The student can express an algorithm in a language.
- 21: The student can evaluate a program for correctness.
- 22: The student can develop a correct program.
- 23: The student can employ appropriate mathematical and logical concepts in programming.

MORE ALGORITHMS

THE STATE GUESSING GAME

- The Rules
 - I will choose a state
 - I will answer questions in such a way that my answers to your questions are true/false (yes/no)
 - You may have up to 5 guesses to guess the state.



Map from www.infoplease.com

SEARCHES

- The strategy of trying to divide choices in half, then again and again until the answer is found is called a Binary Search.
 - Remember NoseGuy and the Guessing Game?
- We are going to look at strategies (or Algorithms) for searching for the “correct” answer.

SEARCHING VS. SORTING

- When you search a list, it is to determine if a specified element is present.
 - There are two primary searching algorithms
 1. Linear Search
 2. Binary Search
- Sorting is done to order the values in the list based upon some key value.
 - There are three primary sorting algorithms
 1. Selection Sort
 2. Insertion Sort
 3. Merge Sort

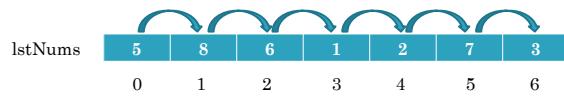
SEARCHING ALGORITHMS

Where is that element?

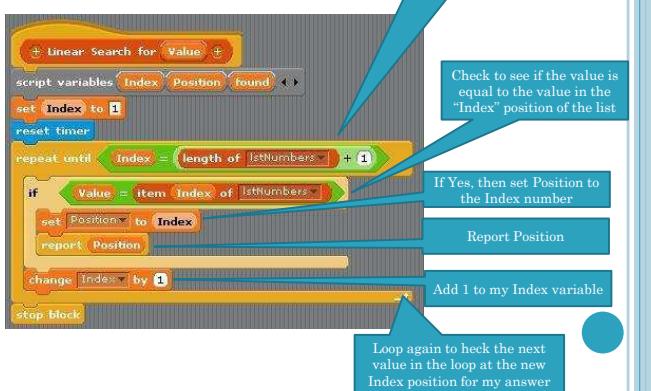
LINEAR SEARCH

- A linear search algorithm searches the list in a sequential manner.

- The algorithm moves through the list, comparing the key value with the values of the elements. If it does not find the key value, it simply moves to the next element.



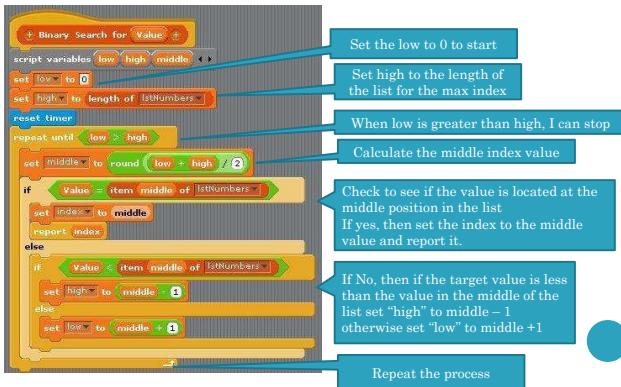
LINEAR SEARCH



BINARY SEARCH

- The binary search algorithm is more efficient than the linear search algorithm.
- Binary search requires that the array be sorted first.
- The algorithm splits the array and checks the middle value.
 - If it is not found it compares the values.
 - If the search value is higher than the middle value, the algorithm moves to the upper half (now a subarray). (Lower – it moves to the lower half).
 - It splits the subarray in half, checks the middle for a match.
 - If not found, it checks to see if it is higher/lower and moves to appropriate subarray.
 - This continues until it has no more values or finds a match.

BINARY SEARCH



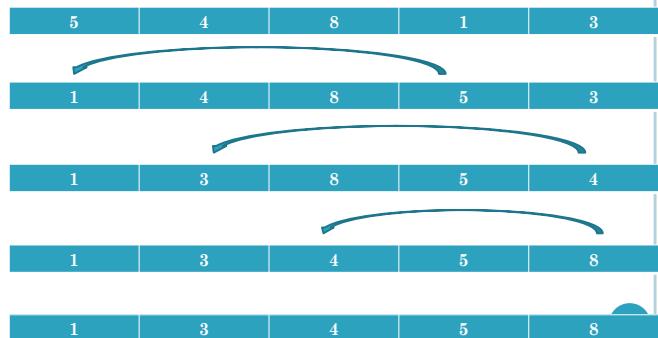
SORTING ALGORITHMS

C# Programming

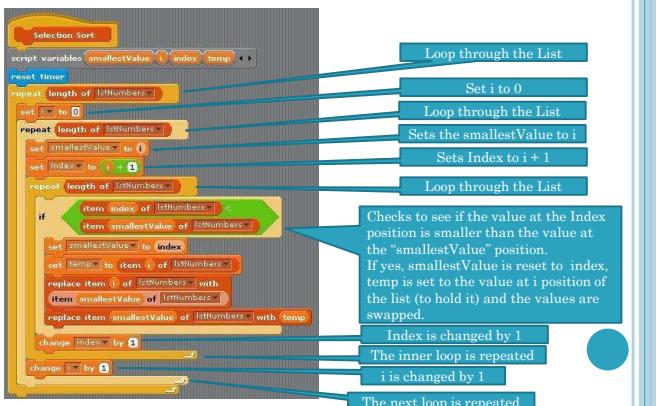
SELECTION SORT

- This is a simple sorting algorithm.
- It moves through the array looking for the lowest value, then moves it to the front.
- The second pass through the array, it looks for the second lowest and moves it to the second position.
- It keeps passing through the array until the last iteration.

SELECTION SORT



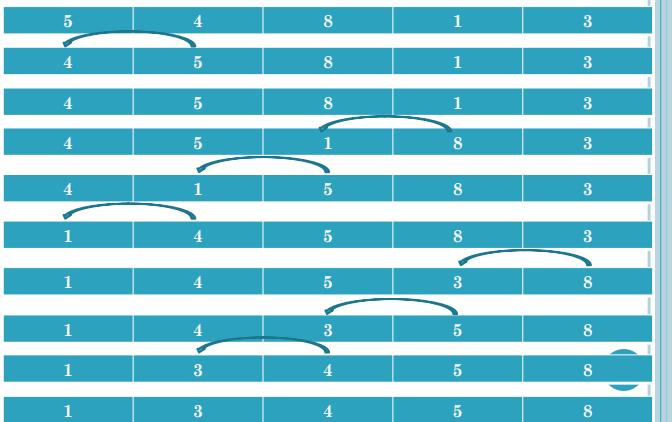
SELECTION SORT



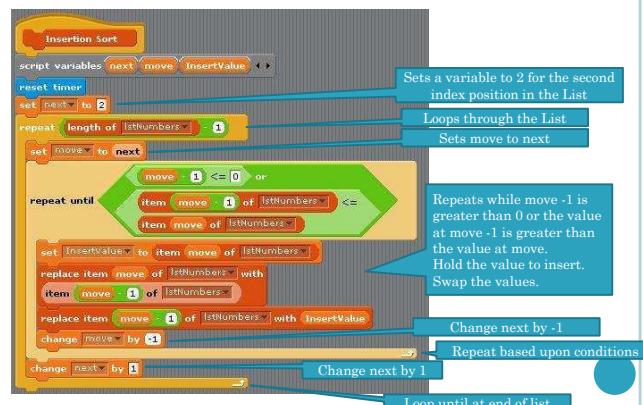
INSERTION SORT

- Another simple sorting algorithm.
- 1st Iteration – Compares element 1 & 2 – Swaps if element 1 > element 2.
- 2nd Iteration – Looks at element 3 – Inserts it in position given element 1 & 2.
- It keeps comparing and inserting until the end.

INSERTION SORT



INSERTION SORT



MERGE SORT

- The merge sort algorithm sorts by splitting the array into two subgroups, sorting the subgroups, then merging them back together sorted.

56	18	65	17	35	29	44
56	18	65	17	35	29	44
56	18	65	17	35	29	44
56	18	65	17	35	29	44
18	56	17	65	29	35	44
17	18	56	65	29	35	44
17	18	29	35	44	56	65

MEASURING EFFICIENCY

- How efficiency an algorithm is can be measured using Big-O notation. (Also called Landau's symbol)
- It tells you how fast a function grows or declines.
- Big-O notation measures the worst-case runtime for an algorithm.

EFFICIENCY OF SEARCHING

- Linear Search
 - Worst Case time $O(N)$ - It goes through entire list
- Binary Search
 - Worst Case time $O(\log N)$ – The number of times N can be divided in half before there is nothing left
 - This is better than the linear search.
- Selection Sort & Insertion Sort
 - Worst Case time $O(N^2)$
- Merge Sort
 - Worst Case time $O(N \log N)$
 - This is a little better than the selection and insertion sorts.

[Curriculum \(/bjc-course/curriculum\)](#) / [Unit 6 \(/bjc-course/curriculum/06-searching-sorting\)](#) /

[Lab 1 \(/bjc-course/curriculum/06-searching-sorting/labs/01-searching-activity\)](#) /

Activity - Searching

We're going to step away from the computer for this activity. First things first: divide up into groups. Let's examine a common computer science problem: finding a number in a list of numbers. We can assume that the user will provide a specific number to look for, and we need to develop an algorithm that will find what location the number they're searching for is located in (or if it's absent). A computer can only check one position in a list at a time, and checking positions takes time. Let's say we're searching through a list where the numbers are random (within a range) and unique (the same number won't be in there twice). As you might imagine, there are several different ways that you can find a particular number in this type of list, but we know that many algorithms have advantages over others. We'll use a set of numbered cups in this activity to represent different pieces of data. Let's call the number of cups N .

Our first goal is to find a specific number among a bunch of randomly chosen cups. Using your set of cups as a testbed, work with your group to **develop an algorithm (remember: set of steps) to do this**. Your algorithm should work for any set of random numbers. **Discuss the following ideas with your group to help your thought process:**

- What strategies exist for finding a particular number in the fewest number of location checks?
 - How many guesses would it take on average to find a number?
 - How many guesses would it take to determine that a number didn't exist in the list at all?
 - How much additional work will it be to find a number if N were to increase?
-

Curriculum (/bjc-course/curriculum) / Unit 6 (/bjc-course/curriculum/06-searching-sorting) /

Lab 2 (/bjc-course/curriculum/06-searching-sorting/labs/02-searching-game) /

A (Non-Video) Game

Let's play a number-guessing game (you have already seen the Scratch version in lab 4.01).

Between your partner and yourself, decide who will be person G, the guesser: the other person (person P, for picker) picks a number between 1 and 50. Person G asks person P questions to determine the chosen number; the only answers that person P can give are "yes" or "no". The guesser should try two different strategies to playing this game:

- Guessing in sequence: "Is the number 1?", "Is the number 2?", "Is the number 3?", and so on.
- Guessing halfway: You know the number must be between 1 and 50, so you start off by asking "Is the number greater than 25?". If the answer is "yes", you instantly know that it must be between 26 and 50, so you ask "Is the number greater than 50?"; otherwise, the number must be between 1 and 25, so you ask "Is the number greater than 12?" Keep asking questions involving the halfway point.

Play several (around 3) games: person P should pick a wide variety of numbers, while person G should use each strategy alternately. As you play each game, think about which strategy involves fewer questions. Which strategy would be better if, say, the guessing game involved numbers from 1 to 1000? From 1 to a million? Discuss why one strategy better than the other?

[Curriculum \(/bjc-course/curriculum\)](#) / [Unit 6 \(/bjc-course/curriculum/06-searching-sorting\)](#) /
[Lab 2 \(/bjc-course/curriculum/06-searching-sorting/labs/03-update-guessing-game\)](#) /

Lab: Add Binary Search to the Guessing Game

Using the starting code provided in lab 5.01, upload your expanded guessing game.

Your Noseguy sprite should utilize binary search in his Guess function to guess the number in as few guesses as possible.

The best route for this would be to set a maximum and minimum value, and adjust them as Noseguy is told “higher” or “lower” by BYOB. Think how you would play as a human. You should have Noseguy able to guess the number in 4 guesses or less.

Unit 7: Recursion - It's not a bad word!

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 4: The student can use programming as a creative tool.
- 20: The programs can use abstraction (procedures) to manage complexity in a program.

Readings/Lectures

- Reading 7.01: All Algorithms are Not Equal ([/bjc-course/curriculum/07-recursion/readings/01-what-is-recursion](#))
- Reading 7.02: Algorithm Analysis ([/bjc-course/curriculum/07-recursion/readings/02-recursion-in-a-nutshell](#))

External Resources

- Berkeley Lecture on Recursion (<https://coursesharing.org/courses/6/lectures/17>)
- Koch's Snowflake, Explained (<http://math.rice.edu/~lanius/frac/koch.html>)
- Snowflake BYOB File ([/bjc-course/curriculum/07-recursion/code/snowflake.ypr](#))

Labs/Exercises

- Lab 7.01: Make a Recursive Tree ([/bjc-course/curriculum/07-recursion/labs/01-make-a-recursive-tree](#))
 - Lab 7.02: Recursive Bunnies ([/bjc-course/curriculum/07-recursion/labs/02-recursive-bunnies](#))
-

What is Recursion?

This activity addresses the concept of **recursion**. It provides an illustration of how recursion is used in computing.

Introduction

What is recursion? A recursive process is one in which objects are defined in terms of other objects of the same type (Wolfram MathWorld). So what does that mean??? It is a function (procedure) that calls itself! When performing recursion, we try to reduce a large problem into a series of similar, smaller problems that can be solved in a similar ways. In fact, it is often best if we can reduce the large problem into a collection of trivially small problems.

The best way to understand is to look at an example. Factorials are an excellent example.

$5!$ (factorial) is $5 * 4 * 3 * 2 * 1 = 120$, right?

So basically...

$5!$

is the same as

$5 * 4!$

Which is the same as

$5 * 4 * 3!$,

Which is the same as

$5 * 4 * 3 * 2!$

Which is the same as

$5 * 4 * 3 * 2 * 1!$

$1!$ is just 1.

Let's look at it in another way: $n!$ is $n * (n - 1)!$ This is what we just demonstrated. If I create a procedure called Factorial that will get the answer of the factorial of the number I send over to the procedure, then I could create a recursive function like this...

$\text{Factorial}(n) = n * \text{Factorial}(n - 1)$

The $\text{Factorial}(n - 1)$ would keep executing until it reached a stopping point, in this case 1 as that is the last number I will need to multiple. So I am using my procedure to "call" itself, sending over a simpler version (in this case a smaller number) until I get to the smallest number (in this case the number 1). This smallest number is what is called the Base Case – when the function should stop calling itself.

$\text{Factorial}(5)$

$5 * \text{Factorial}(4)$

$5 * 4 * \text{Factorial}(3)$

$5 * 4 * 3 * \text{Factorial}(2)$

$5 * 4 * 3 * 2 * \text{Factorial}(1)$

which rolls out like this...

$5 * 4 * 3 * 2 * 1$

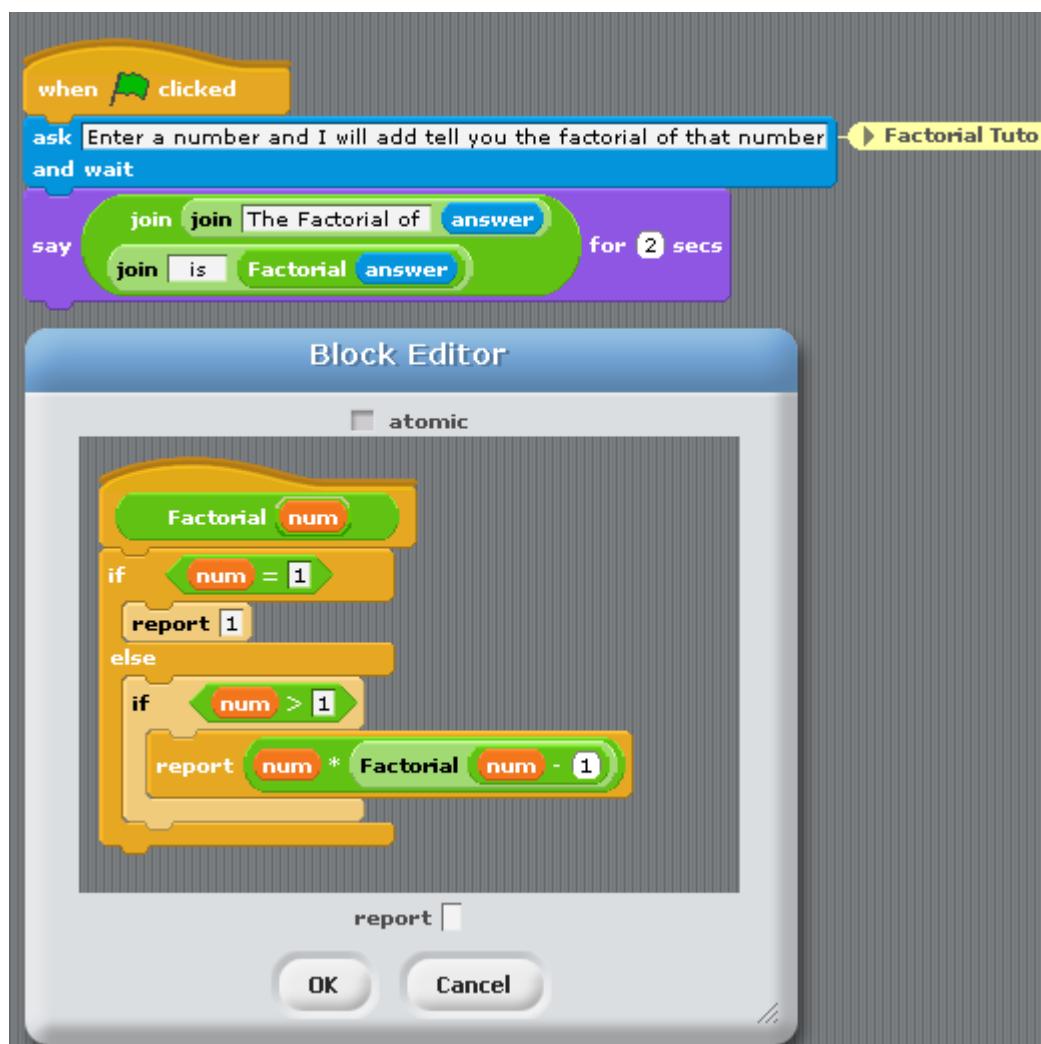
$5 * 4 * 3 * 2$

$5 * 4 * 6$

$5 * 24$

120

Factorial Recursion in BYOB



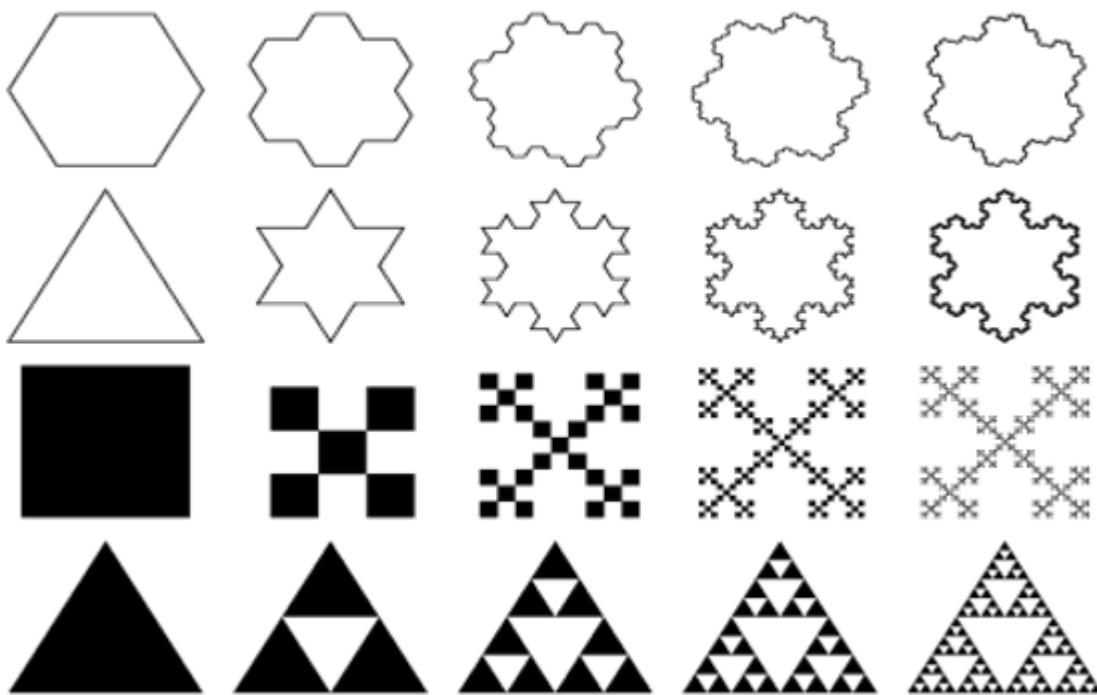
Visual Recursion

Let's look at visual recursion where we have a recursive call that is going to draw something, in our case fractals.

What are Fractals?

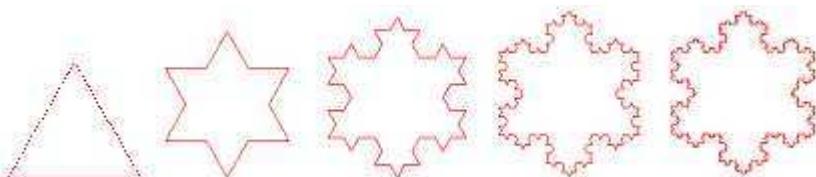
They're everywhere, those bright, weird, beautiful shapes called fractals. But what are they?

Fractals are geometric figures, just like rectangles, circles and squares, but fractals have special properties that those figures do not have. Using your computer, find an image of a fractal. You may see some of the following images.



The Koch Snowflake

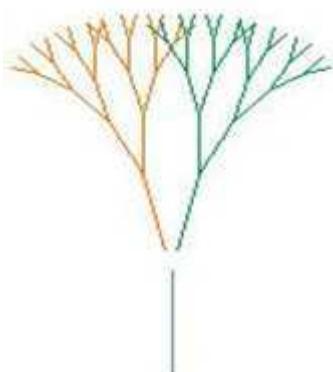
The Koch Snowflake is a famous fractal. How is it made? It starts with an equilateral triangle. Divide one side of the triangle into three equal parts and remove the middle section. Replace it with two lines the same length as the section you removed. Do this to all three sides of the triangle. Keep repeating steps 1, 2 and 3 infinitely and you get the fractal, the Koch Snowflake.



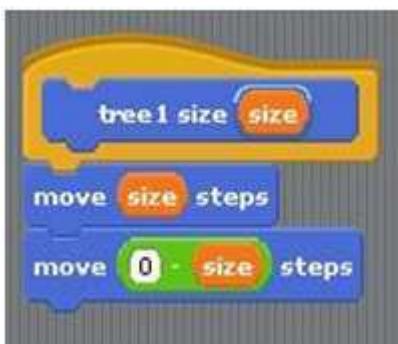
Let's look at another example of recursion.

Recursion to Draw a Tree

This is a simple tree where we want to draw the base (the line for the trunk), then the branches. Note that I am drawing smaller and smaller versions of the same lines. This is perfect for recursion!



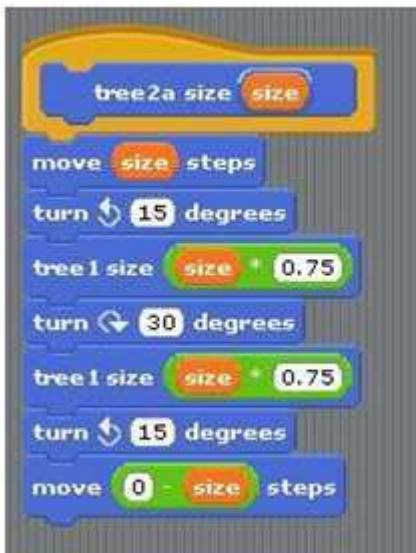
Let's start with drawing the base. This is our "trunk" of the tree.



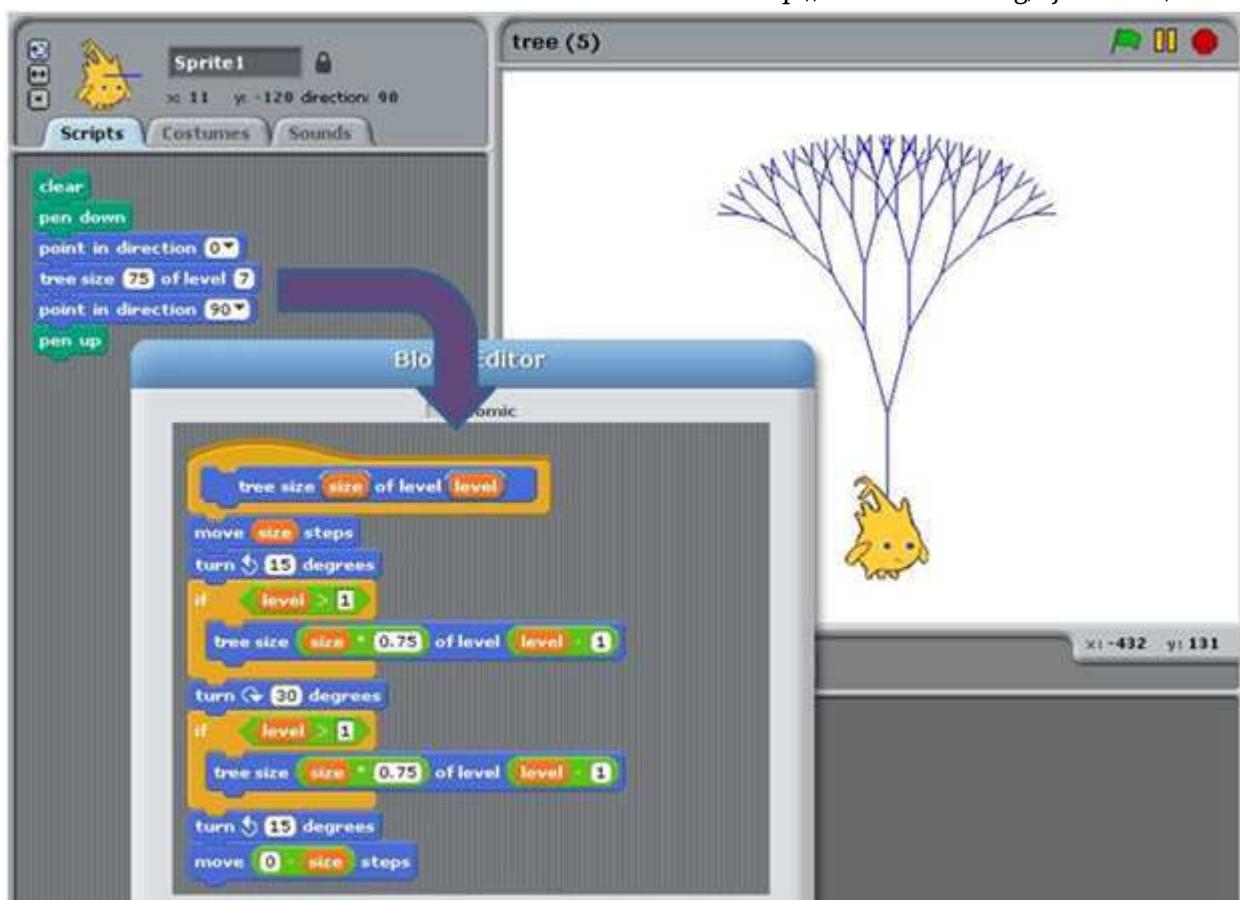
In BYOB, we will tell byob to move “size” steps, then move back. Next, we will need to draw the first branches. Move the steps (back up to the top); then turn and move (75%) and then move (75%) back. Turn back to center + amount to other side; then move (75%) and move (75%) back. This way we get the smaller branches.



We can now use our code from drawing the first tree to help us draw the next level.



We can use variables to make the drawing relative to previous settings. This is one example of the finished code.

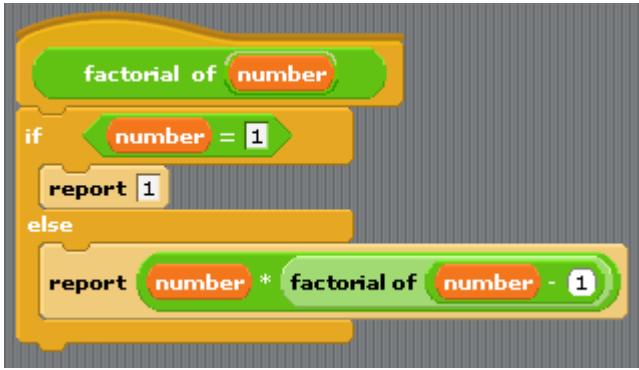


Material from UC Berkeley

Recursion in a Nutshell

We have spent the last couple of labs learning about recursion. You have probably realized that the technique for writing recursive blocks is relatively straightforward, although the body of the block can be a bit tricky to figure out until you have gotten enough practice, or have worked through several examples of what the block should do.

When performing recursion, we try to reduce a large problem into a series of similar, smaller problems that can be solved in a similar ways. In fact, it is often best if we can reduce the large problem into a collection of trivially small problems. For example, finding the factorial (<http://en.wikipedia.org/wiki/Factorial>) of a number is an operation that is often written recursively because each solution builds off of the solution before it ($5! = 4! * 5$; $12! = 11! * 12$). It can be written recursively as a set of trivially small problems (each step only multiplies two numbers) that build up to a significant solution.



The process for writing recursive functions can be summarized in a few steps:

Step 1: Determine the base case

The base case is the simplest possible solution to the overall problem. In the example of the factorial, it is simple to find the answer when the number provided is 1: the answer will always be 1. Every recursive problem has a base case – sometimes it will be straightforward and other times, it may require a little more thought. What would be the base case for the following recursive problems? * Searching through a list for the first occurrence of a particular number and replacing it with zero.

- Counting the number of elements in a list.
- Computing X^Y , where X and Y can both be specified as arguments to the block.

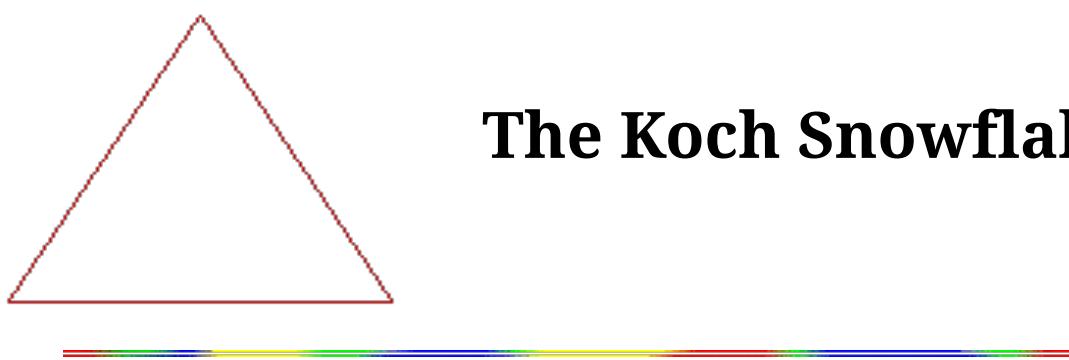
Step 2: Choose a more complicated example

Continuing our work on the factorial problem, we will consider **factorial of 10**. We could choose any number here: 3, 5, 100, 24481, as long as it is not the base case. Hint: sometimes choosing values right next to the base case (2, in this case) makes it harder to detect the relationship in step 3, but any value outside of the base case will do.

Step 3: Consider an example that is slightly simpler (closer to the base case)

factorial of 9 might work here. Assuming we can get the answer to **factorial of 9**, how can we use its solution

to help us figure out **factorial of 10**? Figuring this out will allow us to determine the “relationship” between solutions, which is a pivotal point in writing a recursive function. In this case, $10! = 9! * 10$, which shows a clear relationship that we can represent in BYOB.

Cynthia Lanius

The Koch Snowflake

Table of Contents**Introduction****Why study fractals?**[What's so hot about fractals, anyway?](#)**Making fractals**[Sierpinski Triangle](#)[Using Java](#)[Math questions](#)[Sierpinski Meets](#)[Pascal](#)[Jurassic Park Fractal](#)[Using JAVA](#)[It grows complex](#)[Real first iteration](#)[Encoding the fractal](#)[World's Largest](#)[Koch Snowflake ✓](#)[Using Java](#)[Infinite perimeter](#)[Finite area](#)[Anti-Snowflake](#)[Using Java](#)**Fractal Properties**[Self-similarity](#)[Fractional dimension](#)[Formation by iteration](#)**For Teachers**[Teachers' Notes](#)[Teacher-to-Teacher](#)**Comments**[My fractals mail](#)[Send fractals mail](#)**Fractals on the Web**

You may print and use this [triangle grid paper](#) to help you with this drawing.

Step One.

Start with a large equilateral triangle.

Step Two.

Make a Star.

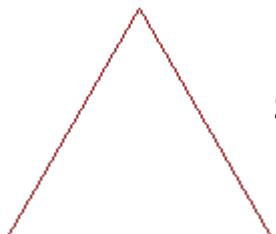
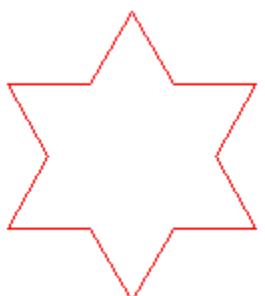
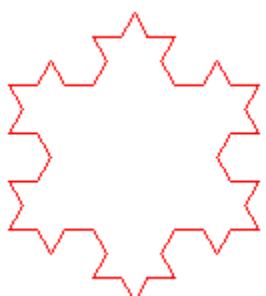
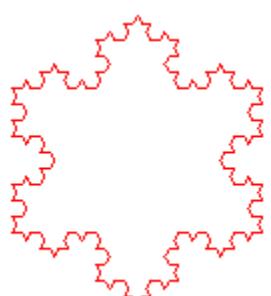
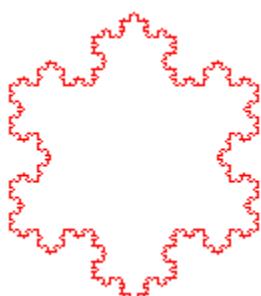
1. Divide one side of the triangle into three equal parts and remove the middle section.
2. Replace it with two lines the same length as the section you removed.
3. Do this to all three sides of the triangle.

Do it again and again.

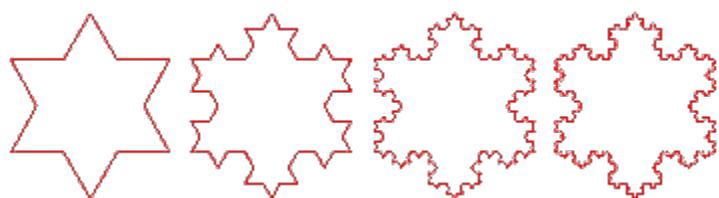
Do it infinitely many times and you have a fractal.

Want to take a long, careful look at what it looks like?

See a few of the steps below.

[The Math Forum](#)**Other Math Lessons**[by Cynthia Lanius](#)**Step One****Step Two****Step Three****Step Four****Step Five**

Let's see them all together



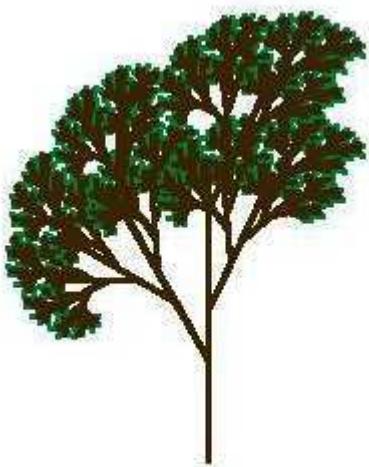
Copyright 1996-2007 Cynthia Lanius

URL <http://math.rice.edu/~lanius/frac/koch.html>

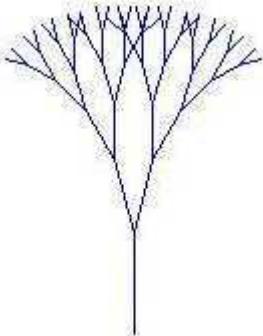
Make a Recursive Tree

This assignment must be done using the BYOB (Build Your Own Blocks) version of BYOB/SNAP. When you make blocks, **be sure that the ATOMIC box is not checked!** (This will allow you to stop your program if you write a program that will go on forever!)

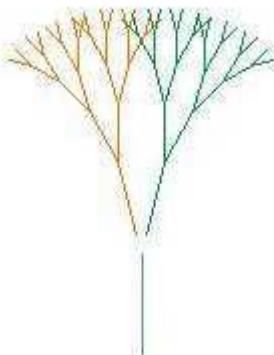
Our goal is to draw a tree like this:



but we'll start with a simpler version that shows the technique clearly, although less prettily:



The key to understanding this technique is to see that the tree is a fractal, that is, a picture that contains smaller versions of itself:



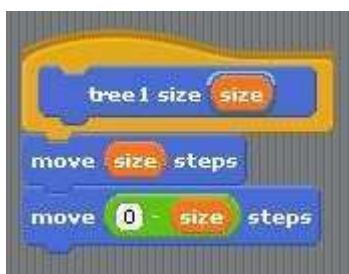
We're going to create a TREE block in BYOB BYOB/SNAP. It'll start with a MOVE block to draw the trunk of the tree,

then a TURN block turning left, then a TREE block to draw the left smaller tree, and so on.

"Wait!" you're probably thinking. "How can we use a TREE block inside a TREE block? It doesn't exist yet!" That's the big idea for this assignment.

We're going to work up to the complicated tree starting with very simple steps.

Step 1. Make a TREE1 block (so named because it draws just one level of the tree) using this script:

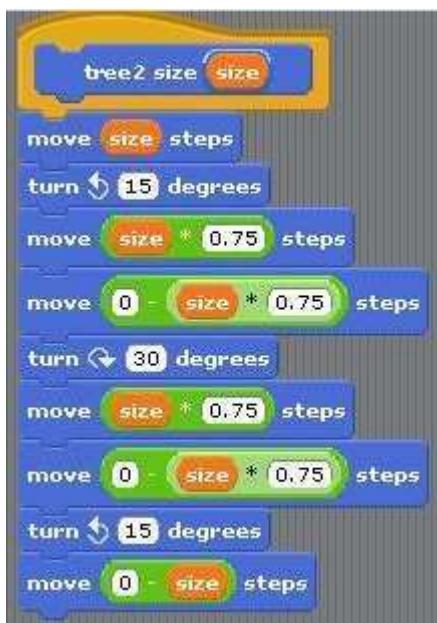


This looks ridiculously simple, but trust us, it'll get interesting soon. When run, the script draws one tree branch, and then *moves the sprite back to its original position*. That's going to be really important when we start using scripts within scripts; we always want to be able to assume that our tree blocks leave the sprite in the same position, and facing the same direction, after it as before it.

Step 2. Point the sprite facing upward, and put the pen down. Then try TREE1 SIZE 50. You should get a result something like this:



Step 3. Make a TREE2 block that draws two levels:

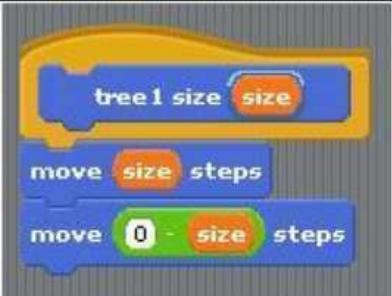
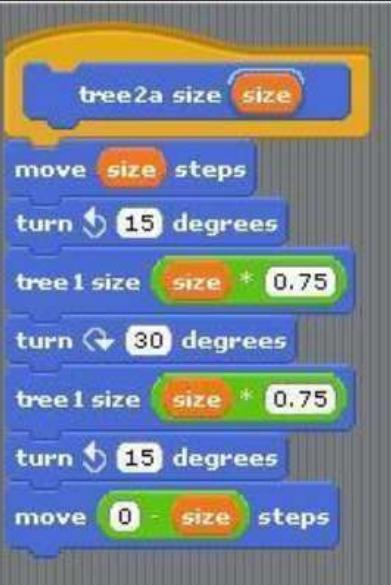


When run, it should give this result:



- At this rate we'll never be able to make the beautiful version of the tree. In the next step, we'll discuss how to simplify this process.
- Before going on, make sure that you can mentally trace through the code provided. Paying close attention to the forwards and turns is going to be important to see the big picture and understand the recursion.

Step 4. That TREE2 block is pretty long and repetitive. But we can simplify it if we notice that in two places it has a move forward/move back pair of blocks, and that this is what TREE1 does! So we can use TREE1 to shorten TREE2. Compare the code below to convince yourself that the new Tree2a will work the same as the original Tree2.

Tree1	Old Tree2	New Tree2a
		

Note that the tree blocks inside this TREE2A script are TREE1 blocks, not TREE2 blocks! So there's no mysterious TREE-using-TREE situation here; it's not unusual for one block to be used in another block's script.

Step 5. Make a TREE3 block that uses the TREE2 block, on the same pattern, and see if you get the result that you expect.

Step 6. If you can stand it, make a TREE4 block that uses the TREE3 block and try it out.

This would be a good time to save your project.

Step 7. Okay, here's the big idea: We can write a TREE block that uses itself in its own script *provided that it knows how many levels it's expected to draw!* So, in addition to the size input, it'll have a LEVELS input:

In the earlier steps, TREE3 used TREE2; TREE2 used TREE1. Here, TREE will use TREE, but reducing the number of levels by 1.

Step 8. Once you can draw a tree of five or six levels using your TREE block, see if you can make one like the first picture in this handout. It's different from what we've done so far because the smaller trees are drawn part way up the trunk, instead of at the top of the trunk, and because the pen color is green for the lowest-level branches (the

TREE1-like ones) and brown for the others. You don't have to get it exactly like the picture; just try to make a more realistic-looking tree.

Vary Your Tree

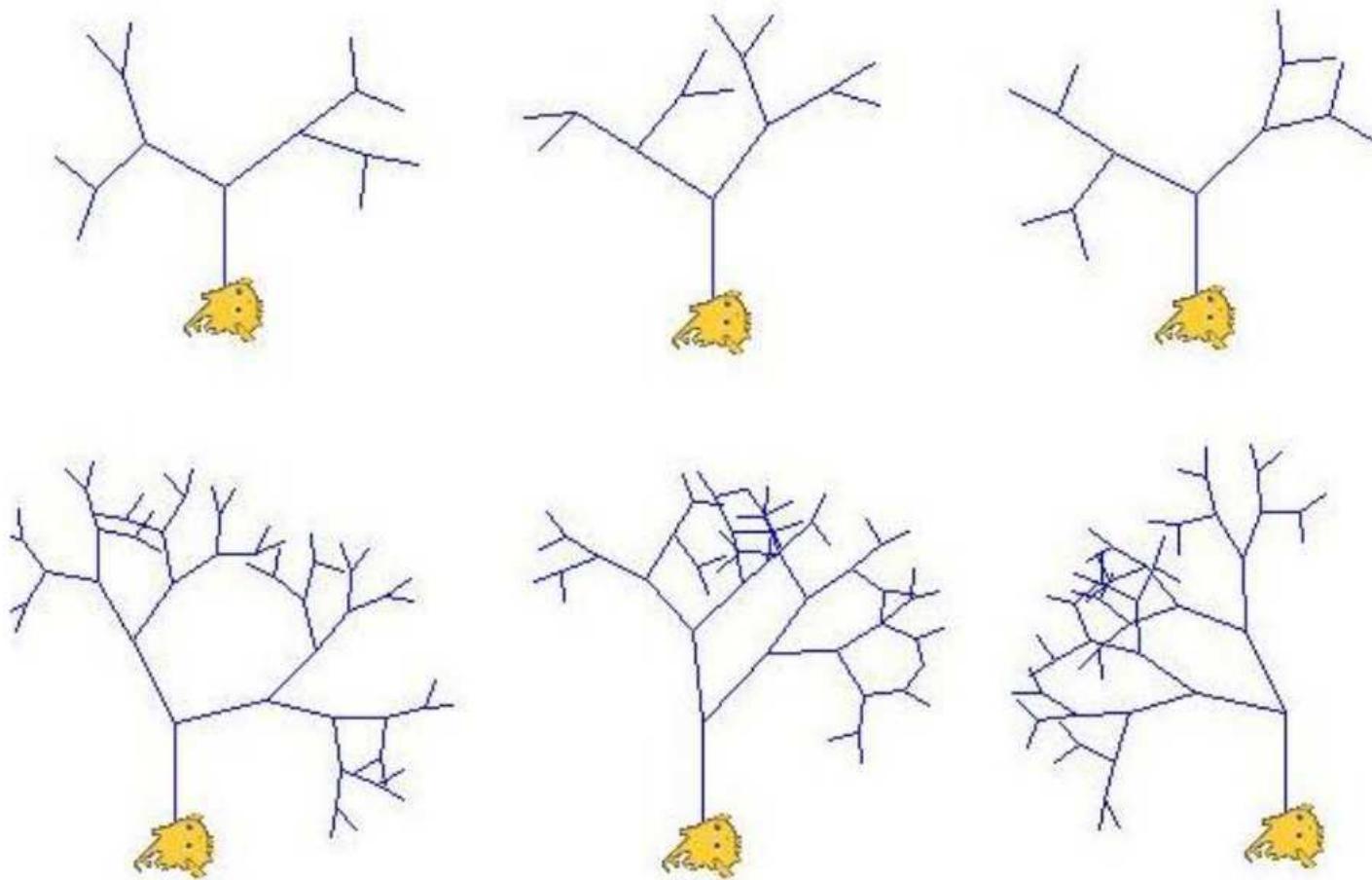
Change your tree in the following ways. You should probably try these one at a time so that you can always tell which change you made caused a new bug.

- change the turn angle
- change the scale factor
- change the number of recursive calls

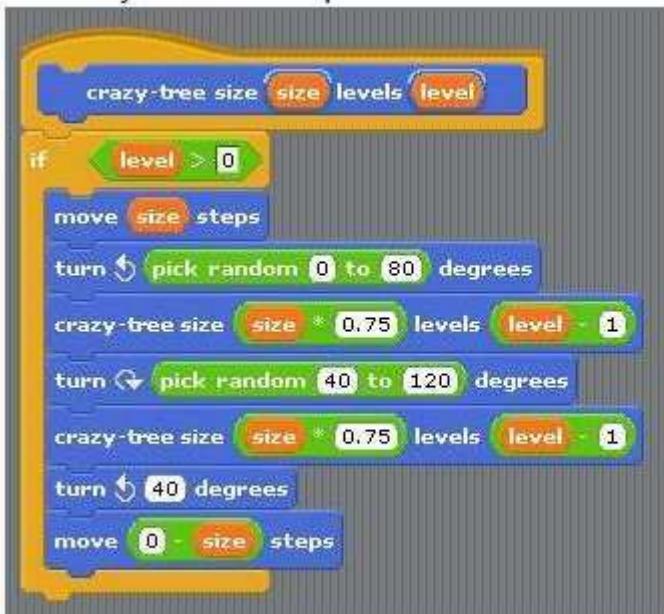
Note: If your character goes off the screen your image might get messed up because you'll tell your character to change their y-position but the character literally won't be able to.

Random Tree

We want to make a tree that looks more random. Here are some pictures of different runs of the program with different levels so that you can see what we were thinking.



Unfortunately our attempt (shown below) was unsuccessful (see "tree" on the right). Modify the crazy-tree code to produce the random trees as shown above.



Base Cases

Now that you have some experience with some recursion, we're going to point out some of the anatomy of a recursive procedure.

We always need a way to figure out that we're done and shouldn't call the recursive call again. Lines 2-4 are the "Base Case" where we handle things that are SO simple we don't need to call the recursive case again.

If this isn't the base case we think to ourselves "Woah - That seems complicated. I'll just do a small part of the problem and delegate to someone else". Here's what we do:

- draw the trunk of the tree (line 6),
- position the character for the left sub-tree (line 7),
- delegate to another copy of the tree block to draw the left sub-tree (line 8),
- re-position the character for the right sub-tree (line 9),
- delegate to another copy of the tree block to draw the right sub-tree (line 10),
- re-position the character to retrace the trunk (line 11), and
- re-trace the trunk and leave the character exactly where it started out.

This wasn't the "best" one we discussed in a previous step, but is a good example of a base case. (Note here we addressed the problem of getting infinite recursion when we have a level less than 0.)

```
1- tree size size levels level
2- if level < 1
3-   move size steps
4-   move 0 - size steps
5- else
6-   move size steps
7-   turn ← 15 degrees
8-   tree size size * 0.75 levels level - 1
9-   turn ↵ 30 degrees
10-  tree size size * 0.75 levels level - 1
11-  turn ← 15 degrees
12-  move 0 - size steps
```

Recursive Bunnies

Use BYOB to create a recursive procedure to count the number of bunny ears given a number of bunnies entered. Use the Factorial or OddNumSum program as a guide. Remember, there are two ears per bunny (unless you have “different” bunnies).

Your code MUST be recursive - You should have a function (block) to count the ears that calls itself!

HINT

Looking at Factorial in our example and OddNumSum we can see that our recursive call should look something like:

The Number of Ears + BunnyEars (Number of Bunnies - 1) where BunnyEars is my recursive function block name and I have a variable to represent “The TOTAL Number of Ears” and the “Number of Bunnies” and the “Number of Ears” would be 2...

because we want to add 2 (or 3) ears to the total of the smaller group of bunnies. Right?

EXTRA POINTS

To earn extra points modify your script code so that:

The ODD number bunnies have 3 ears and the EVEN number bunnies have 2 ears.

For example: I have 5 total bunnies. This means that bunnies #1, 3, and 5 have 3 ears each for a total of 15 ears and bunnies # 2 & 4 have 2 ears each for a total of 4 ears. So my output would say “There is a total of 19 ears.”

By the way, you can have 2 recursive calls inside your script as long as there is conditional execution. In other words, If _____ do the recursive call this way Else do the recursive call that way.

Turn in Your Work

- Create a page titled “Recursive Bunnies” under your Classwork category and describe your logic in creating your recursive procedure.
- Upload your Recursive Bunnies file (.ypr)
- Add the Screenshots to a Word doc with your description. Submit on Moodle.

Unit 8: In the Clouds

Cloud Computing, Security & Encryption

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 4: The student can use programming as a creative tool.
- 5: The student can describe the combination of abstractions used to represent data.
- 9: The student can use models and simulations to raise and answer questions

Readings/Lectures

- Reading 8.01: What is Cloud Computing? (/bjc-course/curriculum/08-cloud-computing/readings/01-what-is-cloud-computing)
- Worksheet 8.02: Hand Biometrics (/bjc-course/curriculum/08-cloud-computing/readings/02-hand-biometrics-worksheet)

External Resources

- The History of Encryption (<http://visual.ly/history-encryption>)
- The Enigma War, 1939-1942 (<http://www.turing.org.uk/scrapbook/ww2.html>)
- Who was Alan Turing? (<http://www.cs4fn.org/magazine/magazine14.php>)
- Locking a Dead Man's Chest (<http://www.cs4fn.org/binary/lock/>)
- How to control a computer with a banana (<http://www.cnn.com/2013/04/05/tech/innovation/jay-silver-makey/index.html>)

Labs/Exercises

- Lab 8.01: Encryption and Decryption (/bjc-course/curriculum/08-cloud-computing/labs/01-encryption-decryption)
- Project 8.02: Encryption Project (/bjc-course/curriculum/08-cloud-computing/labs/02-encryption-project)

What is Cloud Computing?

Much of our work this semester will be supported by software that is provided as a *service* rather than as a *product*. This is example of cloud computing (http://en.wikipedia.org/wiki/Cloud_computing).

For example, the software you used to create your portfolio is an example of cloud computing. Setting up the portfolio used software that was provided as a free service by Google. Your portfolio lives on the cloud, as the Internet is popularly called. Basically, you don't know exactly where your portfolio page is stored, as you would if it were a product (like Microsoft Word) on your laptop.

Cloud Computing Paper

Look up up cloud computing on Wikipedia (http://en.wikipedia.org/wiki/Cloud_computing) and read enough of that article to get a sense of cloud computing and to be able the questions below.

In your summary answer the following questions about cloud computing:

- Cloud computing can involve various services, including computation, data storage, and data access. Which of these services are provided in the case of your portfolio? Explain.
- What are one or two of the main benefits of cloud computing?
- What are one or two of the possible drawbacks of cloud computing?

Material from Dr. Ralph Morelli, Trinity College

Hand Biometrics Technology

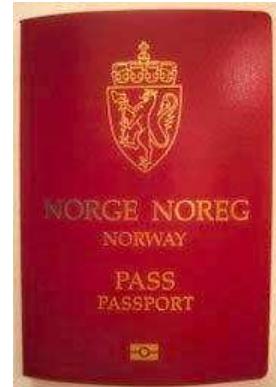
Student Worksheet:



Biometrics (ancient Greek: bios = "life", metron = "measure") is the study of methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits. In information technology, "biometric authentication" refers to technologies that measure and analyze human physical and behavioral characteristics for authentication purposes. Examples of physical (or physiological or biometric) characteristics include fingerprints, eye retinas and irises, facial patterns and hand measurements, while examples of mostly behavioral characteristics include signature, gait and typing patterns.

Sample Applications

1. Since the beginning of the 20th century, Brazilian citizens have used ID cards that incorporate fingerprint-based biometrics.
2. Microsoft has introduced a fingerprint reader that prevents computers from being used by unauthorized people.
3. Some countries have implemented biometric passports that combine paper and electronic identity -- using biometrics to authenticate the citizenship of travelers. The passport's critical information is stored on a tiny RFID computer chip. The icon is incorporated onto most biometric passports to indicate the technology.



Hand Geometry Biometrics

Hand geometry is a biometric that identifies users by the shape of their hands. Hand geometry readers measure a user's hand along many dimensions and compare those measurements to measurements stored in a file.

Viable hand geometry devices have been manufactured since the early 1980s, making hand geometry the first biometric to find widespread computerized use. It remains popular; common applications include access control and time-and attendance operations.

Since hand geometry is not thought to be as unique as fingerprints or retinas, fingerprinting and retina scanning remain the preferred technology for high-security applications. Hand geometry is very reliable when combined with other forms of identification, such as identification cards or personal identification numbers. In large populations, hand geometry is not suitable for so-called one-to-many applications, in which a user is identified from his biometric without any other identification.

Hand Biometrics Technology

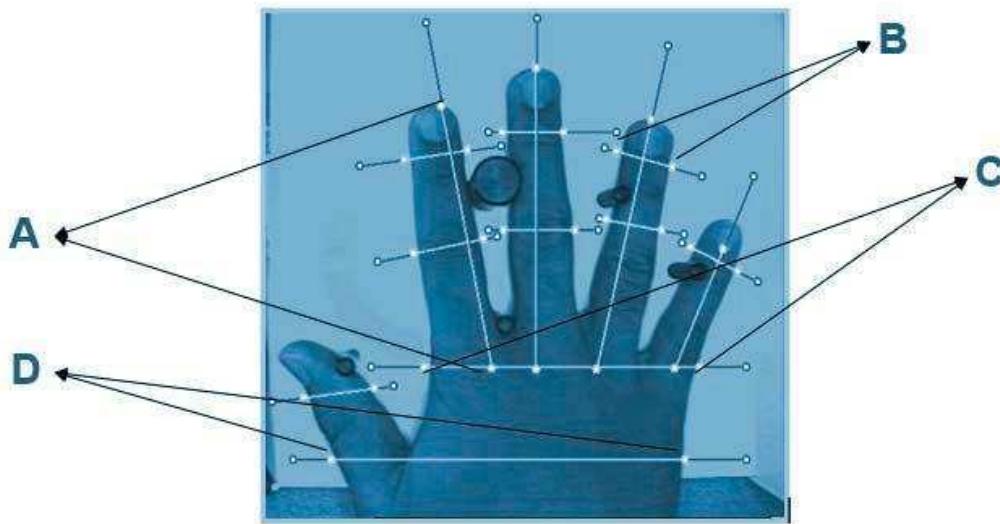


Student Worksheet:

Biometric templates contain information extracted from biometric traits. The resulting codes can be used for identification in a variety of situations. In this activity, you'll determine your own personal hand geometry code.

Step One:

1. Trace your right hand on a piece of paper, keeping the pencil as close to your skin as possible.
2. Using a ruler, measure the following in centimeters (see diagram below):
 - a. Distance from index fingertip to bottom knuckle _____ cm
 - b. Width of ring finger, measured across the top knuckle _____ cm
 - c. Width of palm across 4 bottom knuckles _____ cm
 - d. Width of palm from middle knuckle of thumb across hand _____ cm



3. Record the 4 numbers in A, B, C, D order, which is your personal hand geometry code:
_____ - _____ - _____ - _____

Step Two:

Have someone else in your class trace your right hand, and repeat the measurements above. Record the 4 numbers in A, B, C, D order...are there any differences?
_____ - _____ - _____ - _____

(Note: Biometric information on this page is provided by and used with the permission of The National Biometric Security Project (NBSP). Duplication is permitted for educational purposes only.)

Hand Biometrics Technology

Developed by IEEE as part of TryEngineering
www.tryengineering.org

Hand Biometrics Technology

Student Worksheet:



You are a team of computer engineers meeting to determine whether personal hand geometry templates or numbers would be unique enough to serve as an element in a new security system for a museum.

Evaluation Phase

As a team (if working with a partner), examine the geometry templates you have received. These will represent the codes of staff that need to access the museum during evening hours to check on the security of a group of priceless paintings. Discuss and answer the following questions to help form your plan for incorporating biometrics into the museum's new security system.

1. How similar were the geometry template codes you examined? What did you observe that was most similar? What did your team determine to be different in the group?

2. What problems do you envision an employee might encounter as they placed their hand in the biometric scanning device?

3. Are there any guidelines your engineering team would recommend regarding either capturing the codes from each employee, or in scanning the employee's hand at the entrance to the museum?

4. Do you think that fingerprint scans would be more effective? Why? Why Not?

Hand Biometrics Technology

Student Worksheet:



Biometrics can be applied to many situations, such as computer login security, employee recognition, time or attendance record systems, and voter identification. As a team of "engineers" describe three other situations where you think engineers should consider incorporating biometrics technology to solve problems.

Please indicate whether any of these situations might warrant a two-level system, where hand biometrics is one of the two levels of verification:

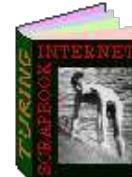
1.

2.

3.

At Walt Disney World, biometric measurements are taken from the fingers of guests to ensure that the person's ticket is used by the same person from day to day. Do you have privacy concerns about this? Why? Why not? If you were part of the engineering team on this project, what would you do to ensure privacy?

The Alan Turing Internet Scrapbook



[Scrapbook Index](#)

Critical Cryptanalysis: The Enigma War, 1939-1942

[Alan Turing Home Page](#) | [Site Map](#) | [Scrapbook Index](#) | [Previous Scrapbook page](#) |
[Next Scrapbook page](#)



The Crown Inn, where Alan Turing lodged
(my photo, 1979)

The Shenley area ([see this map](#))
is now completely changed.

Alan Turing at War

After Britain declared war on Germany on 3 September 1939, British codebreaking operations were moved from London to **Bletchley Park**. This country house was near the then small railway town of Bletchley, half-way between Oxford and Cambridge.

See [this modern map for the site](#), close to Bletchley railway station.

Between 4 September 1939 and the summer of 1944, Alan Turing lodged at The Crown Inn, at **Shenley Brook End**, a village to the west of Bletchley.

War did not stop Alan Turing being an individualist. In 1940 he buried some silver bars near Shenley. In 1944, 1946 and 1952 he tried to find them and failed. No-one knows what happened to his buried treasure!

Bletchley Park today



Bletchley Park Mansion

The **Bletchley Park Trust** website gives full details of how to visit the museum and the various events held there. It has a [history](#) section.

Bletchley Park can still be seen today because the wartime site was left largely unchanged after 1945.

In 1991, the site was saved from property development, and amazing work of reconstruction was done by the original curator, **Tony Sale**, and his collaborators.

Now the house and grounds are managed by the **Bletchley Park Trust**.

The late Tony Sale created a complementary website, www.codesandciphers.org.uk, which still gives extensive technical explanation and copies of original documents.

It also gives a [Virtual Tour of Bletchley Park](#) with many photographs.

See also the [National Museum of Computing](#).

The nerve centre



The huts: my photo, August 1998.

Everything to do with intelligence was dominated by the technicalities of the **Enigma cipher machine**, the key to German communications.

Alan Turing's wartime life was spent mainly in the Huts erected in the grounds of Bletchley Park, where the technical work of codebreaking was done.

Hut Eight, where Alan Turing worked on the naval Enigma, is in the centre of the picture. To the left is Hut Six (Army and Air Force signals). To the right is Hut One. This is where, in 1940, the first Bombe, Turing's codebreaking machine, was installed.

If you want to see an Enigma in real life, [this page](#) by David Hamer tells you some locations. If you are Bill Gates, you may like to [buy one](#).

The Enigma

Most German communications were enciphered on the Enigma cipher machine. It was based on rotors whose movement produced ever-changing alphabetic substitutions.

In its military use, the basic machine was greatly enhanced by a **plugboard**, visible on the front of the machine.

The ciphers it produced were supposed to be unbreakable even by someone in possession of the machine. Ideas of great logical ingenuity were needed to defeat it.



Cheaper: a modern replica.

Even cheaper: [this on-line simulator](#) by Dirk Rijmenants, with elegant and realistic graphical layout and a [challenge](#).

The Enigma has become an icon and a cult object in its own way. This is explored in a book by Dominik Landwehr (in German):

*Mythos
Enigma.*

Who broke the Enigma?

In fact, the Enigma had to be broken afresh *over and over again*. The hardware in the picture is not the whole story, and capturing it did not allow Enigma messages to be read. The German use of the Enigma depended on systems for setting the *keys* for each message transmitted, and it was these *key-systems* that had to be broken. There were many such systems, often changing, and the hardware was changed as well from time to time. The brilliant pre-war work by Polish mathematicians enabled them to read Enigma messages on the simplest key-systems. The information they gave to Britain and France in 1939 may have been crucial, but it was not sufficient for the continuation and extension of Enigma breaking over the next six years. New ideas were essential.

In 1939-40 Alan Turing and another Cambridge mathematician, Gordon Welchman, designed a new machine, the British **Bombe**. The basic property of the Bombe was that it could break any Enigma-enciphered message, provided that the hardware of the Enigma was known and that a plain-text 'crib' of about 20 letters could be guessed accurately.

See the **Enigma report of November 1939**, announcing the British 'superbombe' in production

Alan Turing's mission to France, January 1940, for conference with the Polish analysts



The cottage in the stable yard of Bletchley Park, where Alan Turing worked in 1939-40.
(My photo, July 2002).

Alan Turing made a brilliant contribution to the design with an idea that he himself related to the principle in mathematical logic that 'a false proposition implies any proposition.' It was this idea that overcame the apparently insuperable complication of the plugboard attachment. But that idea was just the beginning of a continuous struggle.

The work done by Turing and his colleagues at Bletchley Park brought cryptology into the modern world. It required ingenious logic, statistical theory, the beginnings of information theory, advanced technology, and superb organisation.



See [Alan Turing's own sketch](#) of the logical principle of his Bombe.

Alan Turing's Bombe

The fullest source of on-line information on the use of the Enigma and the Bombe is on [Tony Sale's site](#).

This gives a full technical description and explanation of the [Enigma machine](#) and its use.

It also offers a '[Virtual Bletchley Park](#)' area with a page detailing his explanation of how [Turing arrived at the idea of the Bombe](#), and another about the [Polish breaking of the Enigma](#) which preceded it.

It includes an on-line [simulator of the Bombe](#), and much more historical information.

A 1944 [American report](#) gives a very clear explanation of how the Bombe was used to break Enigma messages.

Another clear explanation of the principle of the Bombe is given by [Graham Ellsbury](#) of Microvector on his [Enigma pages](#).



The Bombes were built by British Tabulating Machinery at their factory in Letchworth, Hertfordshire.

John Harper describes the work of [rebuilding a copy of Turing's Bombe](#), with much further information about the engineering involved.

[News story on the running of the rebuilt Bombe, 6 September 2006.](#)



Rotors on the [mock-up Bombe](#), June 2001

Totally secret for thirty years

Everything about the breaking of the Enigma cipher systems remained secret until the mid-1970s. Partial accounts then emerged constrained by continuing secrecy about technical matters. Gordon Welchman gave the central principle of the Bombe in describing his own contribution in [The Hut Six Story](#), 1983.

In the mid-1990s virtually everything was released from secrecy and now it is possible for scholars to investigate this fascinating history in considerable detail.

Notable books are:

- [Codebreakers](#), eds. F. H. Hinsley and A. Stripp.
- Ralph Erskine and Michael Smith (eds.) [Action This Day](#)
- Stephen Budiansky [Battle of Wits: The Complete Story of Codebreaking in World War II](#)
- F. L. Bauer, [Decrypted Secrets](#), gives a full account of the



Hut 6, Bletchley Park, where the German Air Force signals were broken with the help of Turing's Bombes. (My photo, 1998)

Enigma as part of a serious work on modern cryptology.

Simon Singh's popular work *The Code Book* claims to explain 'how Turing broke the Enigma.' Unfortunately it makes the Enigma problem look much easier than it actually was, and so undervalues Turing's contribution.

[Mark Baldwin](#) offers illustrated talks and an extensive specialist book service.

Station X

The television programme series *Station X* about Bletchley Park, made for Channel Four television in the UK, was first transmitted in early 1999. The video can be bought on-line from the [Bletchley Park Shop](#). For the United States it was reduced to a two-hour programme shown on Nova as *Decoding Nazi Secrets*. It is evocative and valuable as a archive of interviews but weak in showing that the secret of success at Bletchley Park was the application of scientific method. Michael Smith's book, *Station X*, accompanied the TV series.

Other sites on World War II cryptography

[Frode Weierud's CryptoCellar](#).

[Geoff Sullivan's CryptoMuseum](#)

Alan Turing and the Battle of the Atlantic

The Bombe was used with success from the summer of 1940 onwards, to break messages enciphered on the simpler Enigma system used by the German Air Force. But the most important messages were those to and from the [U-boat fleet](#), and these were enciphered on a much more secure Enigma system.

Alan Turing took on this problem, going against the prevailing view that it would prove unbreakable. Although he had crucial new ideas at the end of 1939, not much practical progress could be made. In 1940 they were desperate.

See the October 1940 [Operation Ruthless](#) plan devised by Ian Fleming, later the creator of 'James Bond', to capture such information for Turing's work.

The breakthrough came in February 1941, with the capture of papers from the *Krebs* off Norway.

From then on, with the help of some further captures, the U-boat communications were effectively mastered. Alan Turing continued to head the cryptanalysis of all German Naval signals in Hut Eight.

The naval Enigma was more complicated than those of the other German services, using a stock of eight rather than five rotors. For the Bombe to work in a practical time it was necessary to find ways of cutting down the number of possibilities. Alan Turing developed 'Banburismus,' a statistical and logical technique of great elegance, to find the identity of the rotors of the enciphering Enigma before using the Bombe. Turing made major developments in Bayesian statistical theory for this work, with his assistant (I. J.) Jack Good.

See this [news article](#) about recent work extending Turing's theory, with [more technical web-page](#) and [downloadable pdf](#) on 'Almost Good Turing.'

See [Steve Hosgood's page](#) on 'Banburismus' for a detailed description of the whole process.

Tony Sale's site also has a sequence of pages on [Naval Enigma](#) explaining in considerable detail what Alan Turing did and how Banburismus worked.

Books concentrating on the naval Enigma capture operations:

- Hugh Sebag-Montefiore, [Enigma: the battle for the code](#). There is a summary of his story on [this naval history page](#)
- David Kahn, [Seizing the Enigma](#)

On 1 February 1942, the U-boats changed to an even more complicated Enigma system, involving a fourth rotor. Their communications became unbreakable until December 1942, when a brilliant trick allowed codebreaking to be restored.

Fact and Fiction

An American film, [U-571](#), drew on the story of the material captured in 1941 but fictionalised it as an American achievement. You can see US Navy comment on this fictionalisation [here](#).

Robert Harris's thriller novel [Enigma](#) has been adapted as a film [Enigma](#) with a screenplay by Tom Stoppard. It depicts the naval Enigma problem in the 1943 period. However the story is fiction, and the film does not show the actual Bletchley Park location. The film also endows the fictional lead character with allusions suggesting he is 'really' Turing himself. See my [review of the film](#).

However, more care was taken in representing the technical background. You can see in [Tony Sale's detailed pages](#) how he scrupulously devised suitable messages and Enigma

methods for this film.

The end of the beginning

At the end of 1942, Alan Turing's war experience had changed him in many ways. It gave him experience of **digital electronics**. It made him fascinated by the idea of '**intelligent machinery**'.

But first, it made him a top level liaison between **the United Kingdom and the United States**.

Continue to the next Scrapbook page.

All these developments were to come together in the post-war world of the computer, with Alan Turing at its centre. But no-one knew it was...

The beginning of the end

CONTINUE: [Next Page](#) | [Previous Page](#) | [Scrapbook Index](#)

Quick
Links:

[Book](#)

[Short Bio](#)

[Scrapbook](#)

[Publications](#)

[Sources](#)

[Alan Turing
Home Page](#)

Search

[Andrew
Hodges](#)

Encryption and Decryption

Learning Objective 27: The student can connect the concern of cybersecurity with the Internet and systems built on it.

- 27a. Identification of tradeoffs associated with the trust model of the Internet.
- 27b. Description of software, hardware, and human components involved in implementing cybersecurity.
- 27c. Explanation of how cryptography is essential to many models of cybersecurity.

Symmetric Key Cryptography

Cryptography means secret writing. **Cryptanalysis** is the study of how to break a secret message, sometimes called a **cryptogram**.

A **cipher** is system for **encrypting** and **decrypting** messages. A cipher's strength – i.e., how secure it is, how well it protects a secret message – depends on the strength and security of its key.

A *symmetric* key cipher is one in which the sender and receiver of the message share the same secret key.

Simple Substitution Cipher

A **simple substitution cipher** is a cipher that is based on a **permutation** of the alphabet. For example,

Plaintext alphabet: abcdefghijklmnopqrstuvwxyz

Cipher alphabet : **ZEBRASROCKDFGHIJLMNPQTVWDX**

There are $26!$ ($= 4.03291461 \times 10^{26}$) ways to permute a 26-letter alphabet – i.e, 26 ways to pick the first letter times 25 ways to pick the second times 24 ways to pick the third, and so on.

In this example, the *symmetric key* is the permuted cipher alphabet. Note how we can use the *keyphrase*, “zebras rock” to construct the alphabet. This makes it easier for two parties to share a secret key.

If Alice wants to send the secret message “attack at noon” to Bob, her fellow agent, she would use the cipher alphabet to encrypt the message, replacing a with Z, t with P, and so on:

Plaintext alphabet: abcdefghijklmnopqrstuvwxyz

Cipher alphabet : **ZEBRASROCKDFGHIJLMNPQTVWDX**

attack at noon

ZPPZBD ZP HIIH

Upon receipt of the message, Bob would use the cipher alphabet in reverse to decrypt the message, replace Z with a, P with t, and so on:

Plaintext alphabet: abcdefghijklmnopqrstuvwxyz

Cipher alphabet : **ZEBRASROCKDFGHIJLMNPQTVWDX**

ZPPZBD ZP HIIH

attack at noon

Breaking a Cipher

Suppose Eve intercepts Alice's message. Eve doesn't know the key. Can she break the secret message?

Question: Would it be practical to try every one of the $26!$ alphabets? No, that would be **intractable**.

If Alice and Bob only sent short messages and changed their key after every message – i.e., **one-time pad** – then Eve will not be able to crack the messages.

But, if they use the key to send message of 40 or more characters, then Eve may be able to break it using **pattern analysis** and **frequency analysis**. How many English words have the pattern 1221 (noon, kook, deed, ...)? If the letter "t" is the second most frequent letter in English, then its substitute, "P", will be the second most frequent letter in the secret message.

Modern Symmetric Cipher

The Advanced Encryption Standard (AES) (http://en.wikipedia.org/wiki/Advanced_Encryption_Standard) is a modern secure electronic cipher used by businesses and the federal government.

AES is an open standard that went through a 5 year public analysis period. AES turns documents into strings of bits and then scrambles and permutes the bits in such a way that it removes all statistical patterns.

What makes AES secure is its large key size, ranging from 128, 192, or 256 bits, which make it **intractable** to use a brute-force search to find the key.

The Key Exchange Problem

How can Alice and Bob share their shared key? It's difficult to image the modern day Internet if the **key exchange problem** had not been solved.

Public Key Cryptography

In **public key cryptography** the key exchange problem goes away because there is no secret key to be shared.

It is based on a **trap door function** or a **one way function** – i.e., a function that is easy to compute in one direction but hard (intractable) to compute in the other.

For example, its easy to multiply $2180 \times 7208 = 15,713,440$. But it would be relatively difficult to factor 15,713,440 into 2180 and 7208.

Simple Public Key Example

Here's a simple example (http://www.di-mgt.com.au/rsa_alg.html#simpleexample) of the Rivest Shamir Adelman (RSA) public key algorithm. It illustrates that we can send a secret message without having to share a secret key.

- Pick two big (100 digit) prime numbers, $p = 11$, $q = 3$
- Let $n = p \times q = 11 \times 3 = 33$
- Let $\phi = (p - 1) \times (q - 1) = 20$

- Choose $e=3$ to be relatively prime to $(p - 1)$ and $(q - 1)$ – i.e., $\gcd(10,3) = 1$ and $\gcd(2,3) = 1$.
- Find $d=7$ such that $\phi(20)$ divides $(3 \times d) - 1$ – i.e., $20 = (3 \times 7) - 1$
- We end up with 3 numbers, n , e , and d that can be used to form Alice's public and private keys:

Alice's **public key**: $(n, e) = (33, 3)$ Alice publishes this key.

Alice's **private key**: $(n, d) = (33, 7)$ Alice keeps this key secret.

Let's suppose Bob wants to send the message "15" to Alice.

Bob encrypts the message using Alice's public key: $153 \bmod 33 = 3375 \bmod 33 = 9$

Alice receives the secret message, "9".

Alice decrypts the message using her private key: $97 \bmod 33 = 4782969 \bmod 33 = 15$

Alice reads Bob's message, "15".

What makes RSA secure is that it's easy to compute $c = m^e \bmod n$, but it's very difficult to compute its inverse – i.e., $m = c^{-e} \bmod n$.

Secure Transactions Across the Internet

In **secure client-server** transactions, RSA is used by the client (Alice) and server (Bob) to exchange a **secret symmetric key**.

Note the role that the **certificate** plays in authenticating the identity of the server.

Questions

- Would it be possible to have today's Internet – Amazon, online banking, etc. – without public key cryptography?
- Can you solve this simple substitution cryptogram?

Qbono kjdo vrp r lcqdbon rq Qncjcqy,

Vbkpo xrpqerhh vrp dhkdgos rq cjxcjcqy.

Kj sryp bo vkths lcqdb,

Qbo erqqonp vkths scqdb,

Pk bcqp vono rj cilppcehcqy.

- Hint: Look at the first words in lines 1 and 3.
- Hint: That first word has a familiar pattern.
- Hint: The last word in the first line has a familiar pattern.
- Here's an online cryptogram tool (<http://www.cs.trincoll.edu/~ram/cryptogrammer/>) to help you solve it. Paste the cryptogram into the tool and then try substituting letters in the alphabet until you get it.

Curriculum (/bjc-course/curriculum) / Unit 8 (/bjc-course/curriculum/08-cloud-computing) /
Lab 2 (/bjc-course/curriculum/08-cloud-computing/labs/02-encryption-project) /

Encryption Project

Using the links provided and your own research on encryption methods used today, create a product about encryption past, present and/or future. You can focus on any aspect of encryption that interests you.

You should have at least 4 sources with one of those not given by me. Make sure to include a Works Cited page to submit with your product.

Work with your table partner to create a product of your choice. The product can be any of the following.

- Wiki/Blog/Web Page
- PowerPoint or Prezi presentation
- Short “Film”
- Voki
- Other by Student/Teacher discussion

TURN IN

- Create a new portfolio page called Encryption.
- Add a description of what you chose to do your project on.
- Upload your product
- Add your Works Cited as a link

SUBMIT IN MOODLE

- Screenshot of your Portfolio Page
- Works Cited document
- Your product (or link if it is on the Internet)

Unit 9: Data, Data, Everywhere

Learning Objectives

- 1: The student can use computing tools and techniques to create artifacts.
- 2: The student can collaborate in the creation of computational artifacts.
- 10: The student can use models and simulations to raise and answer questions.
- The student can use computers to process information to gain insight and knowledge.
- The student can communicate how computer programs are used to process information to gain insight and knowledge.
- The student can use computing to facilitate exploration and the discovery of connections in information.
- The student can use large datasets to explore and discover information and knowledge.
- The student can analyze the considerations involved in the computational manipulation of information.

Readings/Lectures

External Resources

- The Beauty of Data Visualization (http://www.ted.com/talks/david_mccandless_the_beauty_of_data_visualization.html) - TED Talk
- Data Explosion creates Revolution (<http://www.sfgate.com/technology/dotcommentary/article/Web-2-0-Summit-Data-explosion-creates-revolution-2326463.php>) - SFGate
- Big data and society (<http://www.guardian.co.uk/media-network/media-network-blog/2013/apr/12/big-data-privacy-economist>) - interview with Kenneth Cukier, The Economist
- Mike2.0 definition of Big Data (http://mike2.openmethodology.org/wiki/Big_Data_Definition)
- IBM's definition of Big Data (<http://www-01.ibm.com/software/data/bigdata/>)
- The Pandora Music Genome Project (<http://www.pandora.com/about/mgp>)
- GCF Excel Resources (<http://www.gcflearnfree.org/excel2010>)

Labs/Exercises

- Lab 9.01: Top Songs (/bjc-course/curriculum/09-data/labs/01-top-songs)
- Portfolio Project 9.02: Data Portfolio Project (/bjc-course/curriculum/09-data/labs/02-data-portfolio-project)

Resources for Project

- Data Sets (<http://archive.ics.uci.edu/ml/datasets.html>)
- Amazon Data (<https://aws.amazon.com/datasets>)
- Census Data (<https://www.census.gov/main/www/cen2000.html>)
- KDnuggets (<http://www.kdnuggets.com/datasets/>)

© 2013 Hearst Communications Inc.

HEARST newspapers

Big data and society - interview with Kenneth Cukier, The Economist

Big data will transform the world, but issues around privacy and propensity need to be resolved, says Kenneth Cukier

- [Share](#)
- [Tweet this](#)
-  [Email](#)



Big data is something very new and will touch all aspects of society. Photograph: KC
Can you tell us a little bit about your role as data editor for The Economist?

The position is a new one, but it stems from the idea that the new wealth of data and new tools to process and visualise it means that we as journalists can tell stories in new ways. Instead of basing stories on a string of anecdotes with a single statistic dropped in, we can invert the form and make the data the story, while just using a judicious anecdote to illustrate the information. In this respect, [The Economist](#) can be said to have been practising data journalism for 170 years; we're known for our data-driven content like the Big Mac index to compare currencies.

How do you characterise big data?

There is no concrete definition and that is probably a good thing since to define is also to limit. But it's not woolly either. We can understand big data by its features, and the central one is this: we can do things with a huge corpus of data that we are unable to do with smaller amounts, to extract new insights and create new sources of value. This encompasses things like machine learning, in which we have self-driving cars and decent language translation. This is not because we have faster chips or cleverer algorithms, but because we have more data (and the tools to process it at a vast and affordable scale).

In what ways is big data transforming the world?

It is on track to touch all aspects of society. We will go from a world that we understand by experiencing it on an individual level, to one we comprehend on a more universal level. By that, I mean that we tend to base decisions on small amounts of data that are usually just a simulacrum of the complex reality we are trying to deal with, and tailored to our cognitive limitations to make sense of it. Tomorrow, we will use big data to surpass our faith in our individual powers and instead place trust in the data (though not blind trust).

Take medicine. Today, doctors make diagnoses based on their judgement. Sounds reasonable? In time, this will probably be considered as barbaric as bloodletting. Why not use big data? We could enshrine the experience of all doctors, and of hundreds of millions of patients over decades, to identify the best treatments to achieve the best outcomes and spot hidden adverse drug side effects. After all, the sum of all medical knowledge isn't in the possession of any single physician. But if we aggregate vast troves of healthcare information, we may learn what works best, just as Amazon recommends books not based on the inklings of a literary critic but from correlating sales data. This will mark a revolution in how society uses information.

Concerns have been raised regarding the privacy implications of big data. What is your view of this?

Privacy is a big problem today and it will be a bigger problem tomorrow. We need to improve the legal regime to govern privacy to move beyond the system of notice-and-consent (that is, companies inform users what data they collect and how it's used, and people give their okay). In reality it means that people tick a box agreeing to 60 pages of legal jargon with nary a glance. Instead, we need to consider the use and misuse of the data, not just the collection. We need to focus on the area of harm and not just on the inert, potential harm.

Are there any other challenges associated with big data?

While privacy is a problem, a newer issue is 'propensity'. This refers to the idea that algorithms may be making predictions about what we are likely to do, and we may find that we're penalised before we've actually committed the infraction. So big data may assign a 95% likelihood that a certain person will shoplift, or default on a loan, or fail to survive a surgical operation. We'll need to sanctify human agency and freewill. At the same time, we'll need a new class of professional the "algorithmists" to review big data analyses and provide society the same transparency and accountability that we have today, to ensure that big data is not a black box that obviates the public interests.

Finally – what will be your message to the audience at Big Data Week?

Big data is something very new and will touch all aspects of society. Whilst it helps to have the technical skills, the key to success will be in applying one's imagination, creativity, intellectual ambition and risk-taking – characteristics that are intrinsically human and cannot be reduced to number crunching.

Kenneth Cukier will be speaking at Big Data Week London – Putting Data to Work, 25 April, London — visit their website for [further details](#)

Get more articles like this sent direct to your inbox by [signing up for free membership](#) to the [Guardian Media Network](#) – this content is brought to you by [Guardian Professional](#).

Curriculum (/bjc-course/curriculum) / Unit 9 (/bjc-course/curriculum/09-data) /
Lab 1 (/bjc-course/curriculum/09-data/labs/01-top-songs) /

Lab: Top Songs

Learning Objective 13: The student can use large datasets to explore and discover information and knowledge. P3

Evidence for Learning Objective 13: Student work is characterized by:

- Use of large datasets to extract information and knowledge.
- Explanation of how large datasets can facilitate exploration and discovery.

Rolling Stone created a list of the top 500 songs of all time. We've created a .csv file of these files. This file, as well as others you will need, is attached.

Task 1

Determine the top 10 artists based on how many songs the artist has in the top 100. You have a file of the top 100 artists, associated ranking, and song that received that ranking. The artist with the most songs in the top 100 is the top artist. Look at the top 100 artists and songs. Attempt to identify the top artist by looking over the document.

Next, brainstorm how you would approach this computationally.

Task 2

Using a computer, identify the top 10 artists from the top 500. The .csv file (top500.csv) is also available for you. You can do this anyway you want. If it helps we've uploaded the top 500 songs to the IBM Many Eyes website (but you absolutely don't have to use this particular tool). That website lives here:

Top 500 songs in many eyes:

<http://www-958.ibm.com/software/data/cognos/maneyes/datasets/top-500-songs/versions/1>

You can also find these songs by searching data sets on the Many Eyes site for 'song'.

Task 3

Does the approach you developed in the last task work with the data that's the top 1000 rock and roll songs (e.g., as determined by KZOK 102.5)? This data is also attached for you. Spend a couple of minutes looking at it.

Consider the problem of treating all songs equally: the first song and the 500th song count equally in weighting the "top" group by the number of songs in the top 500 (or 1000). What alternatives can you develop to this ranking in determining the "top" group? How would your method of finding the top group change given the alternative you develop?

To Turn In Your Work

Add all of your answers and comments to a document to upload. Make sure to tell how you determined your answers.

This is a test/project grade.

Learning Objectives

The Data Portfolio Task addresses the following CS Principles Learning Objectives (LOs):

- 1: The student can use computing tools and techniques to create artifacts.
- 2: The student can collaborate in the creation of computational artifacts.
- 4: The student can use computing tools and techniques for creative expression.
- 10: The student can use models and simulations to raise and answer questions.

Task

This portfolio entry includes work on a team (usually 2 members) and as an individual. You may work with another student in class if possible.

Collaborative portion:

Identify and describe a significant area in which you will conduct an investigation to gain insight and knowledge from publicly available data. * The area can be an academic topic you are studying or an area of interest.

Develop a set of 3-5 questions that will be the focus of the investigation, find one or more large data sets that will allow you to obtain answers to your questions, and then apply computational tools and techniques to answer your questions, e.g., by finding patterns in the data, by transforming or translating the data, or by finding connections between the data and other sources of knowledge.

- For example, you could find data in Excel spreadsheet format to analyze by using a pivot table or graph/chart.
- Data sets can be tables or spreadsheets. Professional organizations (associations) and government organizations are good places to look for data.

Find data (quantitative data) on the Internet to answer your questions * For example, a data set that you can pull into Excel (or another program).

Produce an artifact (e.g., a report, video, presentation, visualization, or combinations of these) that will allow someone else to understand your investigation, including both the set of questions you pose and the answers you obtain.

The artifact should

- describe the computational tools and techniques you use,
- explanations of why you chose them, and
- why they are appropriate for your investigation.

Individual portion

Create a document in which you

- make it clear that you can use the computational tools and techniques described in your collaborative artifact to verify the answers that are provided there.
- provide enough details about the tools and techniques and their use with the data and other sources of knowledge so that a skilled reader could verify and reproduce your answers using the same data set and other

- sources of knowledge. In other words, you should tell me how you analyzed your data so that I could re-create it.
- This is similar to a lab report in science.

You must include a reflection that describes and analyzes the collaborative aspects of the investigation.

Prepare and submit the following

A collaboratively developed artifact that communicates a detailed description of your group's investigation, the questions, and your collective findings.

- You may use any form of digital artifact (e.g., a report, video, presentation, visualization, or combinations of these) that allows you to best communicate your investigation and findings.
- You and your partner will each submit the same artifact.
- Submit a Works Cited listing all information sources. (MLA format)
- Submit your data file/s.

An individually written document that addresses the investigation.

- Each group member must write her/his own individual document.
- In writing the individual document you must adhere to the Task description above and the Requirements description below in supplying details of your investigation.

Suggestions

- If you are interested in researching different occupations try http://www.bls.gov/oes/current/oes_stru.htm
- If you are interested in researching different colleges try <http://www.cfnc.org/index.jsp>
- There are associations associated with every field, look those up for available data.
- Need help with creating graphs in Excel, try <http://www.gcflearnfree.org/excel2010/17>

Requirements

- You must work with a partner on this project.
- You will collaboratively conduct the investigation: formulating the area to be investigated, developing the questions, finding data set(s) to address those questions, and creating the artifact that allows someone else to understand your investigation.

Rubric

Preparation (20 points)

- Area identified
- 3-5 questions
- Appropriate data set
- Computational tools/techniques chosen

Artifact (40 points)

- Communicates detailed description of investigation
- Questions & answers explained
- Describes tools/techniques used and why chosen/appropriate

Individual (30 points)

- Clear you can use tools/techniques
- Details so that investigation could be duplicated

- Reflection

Works Cited (10 points)

- Appropriate sources
 - MLA format
-

Unit 10: Wonderful World of the Web

The Internet as a System

Learning Objectives

- 27: The student can explain the abstractions in the Internet and how the Internet functions.
- 28: The student can explain characteristics of the Internet and the systems built on it.
- 29: The student can analyze how characteristics of the Internet and systems built on it influence their use.
- 30: The student can connect the concern of cybersecurity with the Internet and systems built on it.
- 31: The student can analyze how computing affects communication, interaction, and cognition.
- 33: The student can connect computing with innovations in other fields.
- 34: The student can analyze the beneficial and harmful effects of computing.
- 35: The student can connect computing within economic, social, and cultural contexts.

Readings/Lectures

- Blown to Bits: Appendix ([/bjc-course/curriculum/10-internet/readings/btb-appendix.pdf](#))
- Reading 10.01: How Google Works ([/bjc-course/curriculum/10-internet/readings/01-how-google-works](#))
- Reading 10.02: Life Before Search ([/bjc-course/curriculum/10-internet/readings/02-life-before-search](#))
- Reading 10.03: Search Takes a Team ([/bjc-course/curriculum/10-internet/readings/03-search-takes-a-team](#))
- Reading 10.04: Blown to Bits Appendix Questions ([/bjc-course/curriculum/10-internet/readings/04-bits-questions](#))

External Resources

- The Rise of Human-Computer Cooperation (http://www.ted.com/talks/shyam_sankar_the_rise_of_human_computer_cooperation.html) - TED Talk
- Packet Switching Demo (http://www.pbs.org/opb/nerds2.0.1/geek_glossary/packet_switching_flash.html) - PBS
- Warriors of the Net Video (http://www.teachertube.com/viewVideo.php?video_id=23140)

Labs/Exercises

- Lab 10.01: How the Internet Works ([/bjc-course/curriculum/10-internet/labs/01-how-the-internet-works](#))
- Lab 10.02: Internet Activity ([/bjc-course/curriculum/10-internet/labs/02-internet-activity](#))
- Portfolio Project 10.03: Internet Portfolio Project ([/bjc-course/curriculum/10-internet/labs/03-internet-portfolio-project](#))

APPENDIX

The Internet as System and Spirit

This Appendix explains how the Internet works and summarizes some larger lessons of its remarkable success.

The Internet as a Communication System

The Internet is not email and web pages and digital photographs, any more than the postal service is magazines and packages and letters from your Aunt Mary. And the Internet is not a bunch of wires and cables, any more than the postal service is a bunch of trucks and airplanes. The Internet is a system, a delivery service for bits, whatever the bits represent and however they get from one place to another. It's important to know how it works, in order to understand why it works so well and why it can be used for so many different purposes.

Packet Switching

Suppose you send an email to Sam, and it goes through a computer in Kalamazoo—an Internet *router*, as the machines connecting the Internet together are known. Your computer and Sam's know it's an email, but the router in Kalamazoo just knows that it's handling bits.

Your message almost certainly goes through some copper wires, but probably also travels as light pulses through fiber optic cables, which carry lots of bits at very high speeds. It may also go through the air by radio—for example, if it is destined for your cell phone. The physical infrastructure for the Internet is owned by many different parties—including telecommunications firms in the

U.S. and governments in some countries. The Internet works not because anyone is in charge of the whole thing, but because these parties agree on what to expect as messages are passed from one to another. As the name suggests, the Internet is really a set of standards for interconnecting networks. The individual networks can behave as they wish, as long as they follow established conventions when they send bits out or bring bits in.

In the 1970s, the designers of the Internet faced momentous choices. One critical decision had to do with message sizes. The postal service imposes size and weight limits on what it will handle. You can't send your Aunt Mary a two-ton package by taking it to the Post Office. Would there also be a limit on the size of the messages that could be sent through the Internet? The designers anticipated that very large messages might be important some day, and found a way to avoid any size limits.

A second critical decision was about the very nature of the network. The obvious idea, which was rejected, was to create a "circuit-switched" network. Early telephone systems were completely circuit-switched. Each customer was connected by a pair of wires to a central switch. To complete a call from you to your Aunt Mary, the switch would be set to connect the wires from you to the wires from Aunt Mary, establishing a complete electrical loop between you and Mary for as long as the switch was set that way. The size of the switch limited the number of calls such a system could handle. Handling more simultaneous calls required building bigger switches. A circuit-switched network provides reliable, uninterruptible connections—at a high cost per connection. Most of the switching hardware is doing very little most of the time.

So the early Internet engineers needed to allow messages of unlimited size. They also needed to ensure that the capacity of the network would be limited only by the amount of data traffic, rather than by the number of interconnected computers. To meet both objectives, they designed a *packet-switched network*. The unit of information traveling over the Internet is a packet of about 1500 bytes or less—roughly the amount of text you might be able to put on a postcard. Any communications longer than that are broken up into multiple packets, with serial numbers so that the packets can be reassembled upon arrival to put the original message back together.

The packets that constitute a message need not travel through the Internet following the same route, nor arrive in the same order in which they were sent. It is very much as though the postal service would deliver only postcards with a maximum of 1500 characters as a message. You could send *War and Peace*, using thousands of postcards. You could even send a complete description of a photograph on postcards, by splitting the image into thousands of rows and columns and listing on each postcard a row number, a column number, and the color of the little square at that position. The recipient

could, in principle, reconstruct the picture after receiving all the postcards. What makes the Internet work in practice is the incredible speed at which the data packets are transmitted, and the processing power of the sending and receiving computers, which can disassemble and reassemble the messages so quickly and flawlessly that users don't even notice.

Core and Edge

We can think of the ordinary postal system as having a *core* and an *edge*—the edge is what we see directly, the mailboxes and letter carriers, and the core is everything behind the edge that makes the system work. The Internet also has a core and an edge. The edge is made up of the machines that interface directly with the end users—for example, your computer and mine. The core of the Internet is all the connectivity that makes the Internet a network. It includes the computers owned by the telecommunications companies that pass the messages along.

An *Internet Service Provider* or *ISP* is any computer that provides access to the Internet, or provides the functions that enable different parts of the Internet to connect to each other. Sometimes the organizations that run those computers are also called ISPs. Your ISP at home is likely your telephone or cable company, though if you live in a rural area, it might be a company providing Internet services by satellite. Universities and big companies are their own ISPs. The “service” may be to convey messages between computers deep within the core of the Internet, passing messages until they reach their destination. In the United States alone, there are thousands of ISPs, and the system works as a whole because they cooperate with each other.

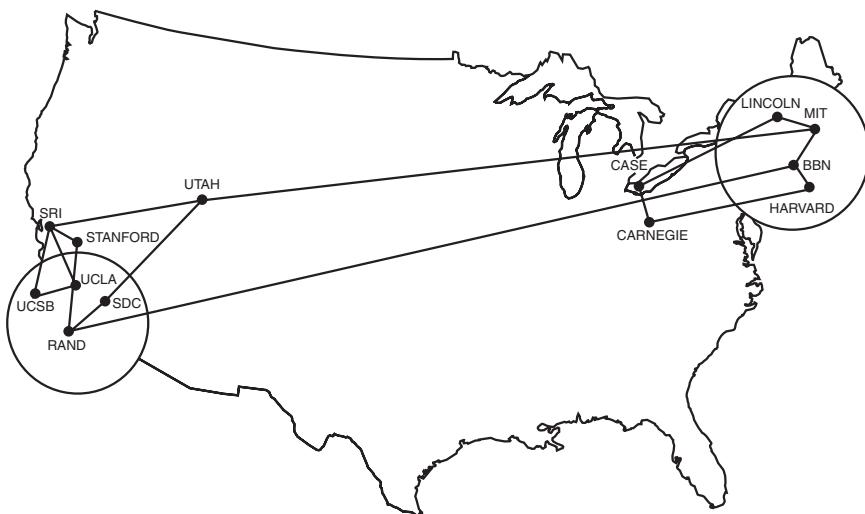
Fundamentally, the Internet consists of computers sending bit packets that request services, and other computers sending packets back in response. Other metaphors can be helpful, but the service metaphor is close to the truth. For example, you don't really “visit” the web page of a store, like a voyeuristic tourist peeking through the store window. Your computer makes a very specific request of the store's web server, and the store's web server responds to it—and may well keep a record of exactly what you asked for, adding the new information about your interests to the record it already has from your other “visits.” Your “visits” leave fingerprints!

IP Addresses

Packets can be directed to their destination because they are labeled with an *IP address*, which is a sequence of four numbers, each between 0 and 255. (The numbers from 0 to 255 correspond to the various sequences of 8 bits,

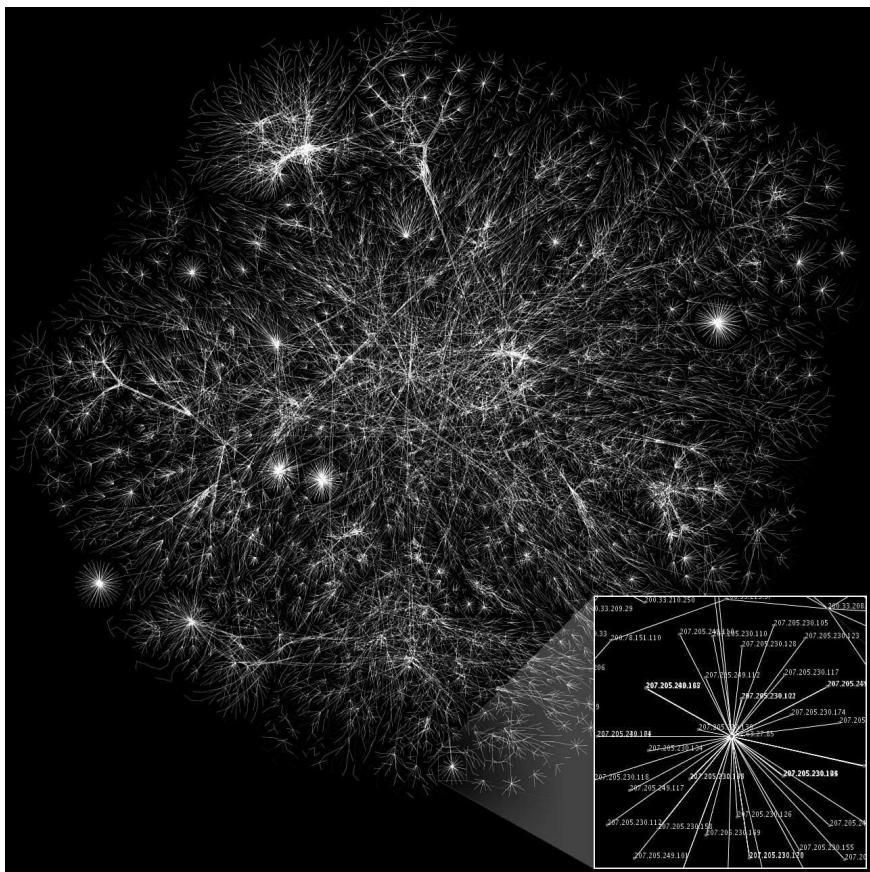
from 00000000 to 11111111, so IP addresses are really 32 bits long. “IP” is an abbreviation for “Internet Protocol,” explained next.) A typical IP address is 66.82.9.88. Blocks of IP addresses are assigned to ISPs, which in turn assign them to their customers.

There are $256 \times 256 \times 256 \times 256$ possible IP addresses, or about 4 billion. In the pre-miniaturization days when the Internet was designed, that seemed an absurdly large number—enough so every computer could have its own IP address, even if every person on the planet had his or her own computer. Figure A.1 shows the 13 computers that made up the entire network in 1970. As a result of miniaturization and the inclusion of cell phones and other small devices, the number of Internet devices is already in the hundreds of millions (see Figure A.2), and it seems likely that there will not be enough IP addresses for the long run. A project is underway to deploy a new version of IP in which the size of IP addresses increases from 32 bits to 128—and then the number of IP addresses will be a 3 followed by 38 zeroes! That’s about ten million for every bacterium on earth.



Source: Heart, F., McKenzie, A., McQuillian, J., and Walden, D., ARPANET Completion Report, Bolt, Beranek and Newman, Burlington, MA, January 4, 1978.

FIGURE A.1 The 13 interconnected computers of the December, 1970 ARPANET (as the Internet was first known). The interconnected machines were located at the University of California campuses at Santa Barbara and at Los Angeles, the Stanford Research Institute, Stanford University, Systems Development Corporation, the RAND Corporation, the University of Utah, Case Western Reserve University, Carnegie Mellon University, Lincoln Labs, MIT, Harvard, and Bolt, Beranek, and Newman, Inc.



Source: Wikipedia, http://en.wikipedia.org/wiki/Image:_Internet_map_1024.jpg. This work is licensed under the Creative Commons Attribution 2.5 License.

FIGURE A.2 Traffic flows within a small part of the Internet as it exists today. Each line is drawn between two IP addresses of the network. The length of a line indicates the time delay for messages between those two nodes. Thousands of cross-connections are omitted.

An important piece of the Internet infrastructure are the *Domain Name Servers*, which are computers loaded with information about which IP addresses correspond to which “domain names” such as harvard.edu, verizon.com, gmail.com, yahoo.fr (the suffix in this case is the country code for France), and mass.gov. So when your computer sends an email or requests a web page, the translation of domain names into IP addresses takes place before the message enters the core of the Internet. The routers don’t know about domain names; they need only pass the packets along toward their destination IP address numbers.

IP ADDRESSES AND CRIMES

The recording industry identifies unlawful music downloads by the IP addresses to which the bits are sent. But an IP address is rarely the exclusive property of an individual, so it is hard to be sure who is doing the downloading. A provider of residential Internet service allocates an address to a home only temporarily. When the connection becomes inactive, the address is reclaimed so someone else can use it. If NAT is in use or if many people use the same wireless router, it can be impossible to establish reliably who exactly used an IP address. If you don't activate the security on your home wireless router, neighbors who poach your home network signal may get you in serious trouble by their illegal downloads!

An enterprise that manages its own network can connect to the Internet through a single gateway computer, using only a single IP address. Packets are tagged with a few more bits, called a “port” number, so that the gateway can route responses back to the same computer within the private network. This process, called *Network Address Translation* or *NAT*, conserves IP addresses. NAT also makes it impossible for “outside” computers to know which computer actually made the request—only the gateway knows which port corresponds to which computer.

The Key to It All: Passing Packets

At heart, all the core of the Internet does is to transmit packets. Each router has several links connecting it to other routers or to the “edge” of the network. When a packet comes in on a link, the router very quickly looks at the destination IP address, decides which outgoing link to use based on a limited Internet “map” it holds, and sends the packet on its way. The router has some memory, called a *buffer*, which it uses to store packets temporarily if they are arriving faster than they can be processed and dispatched. If the buffer fills up, the router just discards incoming packets that it can’t hold, leaving other parts of the system to cope with the data loss if they choose to.

Packets also include some redundant bits to aid error detection. To give a simple analogy, suppose Alice wants to guard against a character being smudged or altered on a post card while it is in transit. Alice could add to the text on the card a sequence of 26 bits—indicating whether the text she has put on the card has an even or odd number of As, Bs, ..., and Zs. Bob can check whether the card seems to be valid by comparing his own reckoning with the 26-bit “fingerprint” already on the card. In the Internet, all the

routers do a similar integrity check on data packets. Routers discard packets found to have been damaged in transit.

The format for data packets—which bits represent the IP address and other information about the packet, and which bits are the message itself—is part of the *Internet Protocol*, or IP. Everything that flows through the Internet—web pages, emails, movies, VoIP telephone calls—is broken down into data packets. Ordinarily, all packets are handled in exactly the same way by the routers and other devices built around IP. IP is a “best effort” packet delivery protocol. A router implementing IP tries to pass packets along, but makes no guarantees. Yet guaranteed delivery is possible within the network as a whole—because other protocols are layered on top of IP.

Protocols

A “protocol” is a standard for communicating messages between networked computers. The term derives from its meaning in diplomacy. A diplomatic protocol is an agreement aiding in communications between mutually mistrustful parties—parties who do not report to any common authority who can control their behavior. Networked computers are in something of the same situation of both cooperation and mistrust. There is no one controlling the Internet as a whole. Any computer can join the global exchange of information, simply by interconnecting physically and then following the network protocols about how bits are inserted into and extracted from the communication links.

The fact that packets can get discarded, or “dropped” as the phrase goes, might lead you to think that an email put into the network might never arrive. Indeed emails can get lost, but when it happens, it is almost always because of a problem with an ISP or a personal computer, not because of a network failure. The computers at the edge of the network use a higher-level protocol to deliver messages reliably, even though the delivery of individual packets within the network may be unreliable. That higher-level protocol is called “Transport Control Protocol,” or TCP, and one often hears about it in conjunction with IP as “TCP/IP.”

To get a general idea of how TCP works, imagine that Alice wants to send Bob the entire text of *War and Peace* on postcards, which are serial numbered so Bob can reassemble them in the right order even if they arrive out of order. Postcards sometimes go missing, so Alice keeps a copy of every postcard she puts in the mail. She doesn’t discard her copy of a postcard until she has received word back from Bob declaring that he has received Alice’s postcard. Bob sends that word back on a postcard of his own, including the serial number of Alice’s card so Alice knows which card is being confirmed. Of course,

Bob's confirming postcards may get lost too, so Alice keeps track of when she sent her postcards. If she doesn't hear anything back from Bob within a certain amount of time, she sends a duplicate postcard. At this point, it starts getting complicated: Bob has to know enough to ignore duplicates, in case it was his acknowledgment rather than Alice's original message that got lost. But it all can be made to work!

TCP works the same way on the Internet, except that the speed at which packets are zipping through the network is extremely fast. The net result is that email software using TCP is failsafe: If the bits arrive at all, they will be a perfect duplicate of those that were sent.

TCP is not the only high-level protocol that relies on IP for packet delivery. For "live" applications such as streaming video and VoIP telephone calls, there is no point in waiting for retransmissions of dropped packets. So for these applications, the packets are just put in the Internet and sent on their way, with no provision made for data loss. That higher-level protocol is called UDP, and there are others as well, all relying on IP to do the dirty work of routing packets to their destination.

The postal service provides a rough analogy of the difference between higher-level and lower-level protocols. The same trucks and airplanes are used for carrying first-class mail, priority mail, junk mail, and express mail. The loading and unloading of mail bags onto the transport vehicles follow a low-level protocol. The handling between receipt at the post office and loading onto the transport vehicles, and between unloading and delivery, follows a variety of higher-level protocols, according to the kind of service that has been purchased.

In addition to the way it can be used to support a variety of higher-level protocols, IP is general in another way. It is not bound to any particular physical medium.

IP can run over copper wire, radio signals, and fiber optic cables—in principle, even carrier pigeons. All that is required is the ability to deliver bit packets, including both the payload and the addressing and other "packaging," to switches that can carry out the essential routing operation.

There is a separate set of "lower-level protocols" that stipulate how bits are to be represented—for example, as radio waves, or light pulses in optic fibers. IP is doubly general, in

IP OVER CARRIER PIGEON

You can look up RFC 1149 and RFC 2549 on the Web, "Standard for the Transmission of IP Datagrams on Avian Carriers" and "IP over Avian Carriers with Quality of Service." They faithfully follow the form of true Internet standards, though the authors wrote them with tongue firmly planted in cheek, demurely stating, "This is an experimental, not recommended standard."

that it can take its bit packets from many different physical substrates, and deliver those packets for use by many different higher-level services.

The Reliability of the Internet

The Internet is remarkably reliable. There are no “single points of failure.” If a cable breaks or a computer catches on fire, the protocols automatically reroute the packets around the inoperative links. So when Hurricane Katrina submerged New Orleans in 2005, Internet routers had packets bypass the city. Of course, no messages destined for New Orleans itself could be delivered there.

In spite of the redundancy of interconnections, if enough links are broken, parts of the Internet may become inaccessible to other parts. On December 26, 2006, the Henchung earthquake severed several major communication cables that ran across the floor of the South China Sea. The Asian financial markets were severely affected for a few days, as traffic into and out of Taiwan, China, and Hong Kong was cut off or severely reduced. There were reports that the volume of spam reaching the U.S. also dropped for a few days, until the cables were repaired!

Although the Internet *core* is reliable, the computers on the edge typically have only a single connection to the core, creating single points of failure. For example, you will lose your home Internet service if your phone company provides the service and a passing truck pulls down the wire connecting your house to the telephone pole. Some big companies connect their internal network to the Internet through two different service providers—a costly form of redundancy, but a wise investment if the business could not survive a service disruption.

The Internet Spirit

The extraordinary growth of the Internet, and its passage from a military and academic technology to a massive replacement for both paper mail and telephones, has inspired reverence for some of its fundamental design virtues. Internet principles have gained status as important truths about communication, free expression, and all manner of engineering design.

The Hourglass

The standard electric outlet is a universal interface between power plants and electric appliances. There is no need for people to know whether their power is coming from a waterfall, a solar cell, or a nuclear plant, if all they want to

do is to plug in their appliances and run their household. And the same electric outlet can be used for toasters, radios, and vacuum cleaners. Moreover, it will instantly become usable for the next great appliance that gets invented, as long as that device comes with a standard household electric plug. The electric company doesn't even care if you are using its electricity to do bad things, as long as you pay its bills.

The outlet design is at the neck of a conceptual hourglass through which electricity flows, connecting multiple possible power sources on one side of the neck to multiple possible electricity-using devices on the other. New inventions need only accommodate what the neck expects—power plants need to supply 115V AC current to the outlet, and new appliances need plugs so they can use the current coming from the outlet. Imagine how inefficient it would be if your house had to be rewired in order to accommodate new appliances, or if different kinds of power plants required different household wiring. Anyone who has tried to transport an electric appliance between the U.S. and the U.K. knows that electric appliances are less universal than Internet packets.

The Internet architecture is also conceptually organized like an hourglass (see Figure A.3), with the ubiquitous Internet Protocol at the neck, defining the form of the bit packets carried through the network. A variety of higher-level protocols use bit packets to achieve different purposes. In the words of the report that proposed the hourglass metaphor, “the minimal required elements [IP] appear at the narrowest point, and an ever-increasing set of choices fills the wider top and bottom, underscoring how little the Internet itself demands of its service providers and users.”

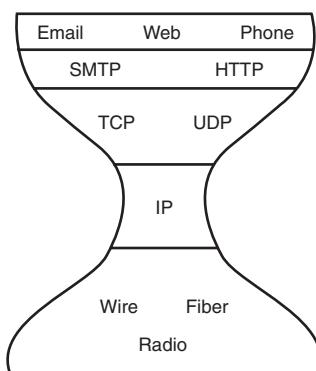


FIGURE A.3 The Internet protocol hourglass (simplified). Each protocol interfaces only to those in the layers immediately above and below it, and all data is turned into IP bit packets in order to pass from an application to one of the physical media that make up the network.

For example, TCP guarantees reliable though possibly delayed message delivery, and UDP provides timely but unreliable message delivery. All the higher-level protocols rely on IP to deliver packets. Once the packets get into the neck of the hourglass, they are handled identically, regardless of the higher-level protocol that produced them. TCP and UDP are in turn utilized by even higher-level protocols, such as HTTP (“HyperText Transport Protocol”), which is used for sending and receiving web pages, and SMTP (“Simple Mail Transport Protocol”), which is used for sending email. Application software, such as web browsers, email clients, and VoIP software, sit at a yet higher level, utilizing the protocols at the layer below and unconcerned with how those protocols do their job.

Below the IP layer are various physical protocol layers. Because IP is a universal protocol at the neck, applications (above the neck) can accommodate various possible physical implementations (below the neck). For example, when the first wireless IP devices became available, long after the general structure of the Internet hourglass was firmly in place, nothing above the neck had to change. Email, which had previously been delivered over copper wires and glass fibers, was immediately delivered over radio waves such as those sent and received by the newly developed household wireless routers.

Governments, media firms, and communication companies sometimes wish that IP worked differently, so they could more easily filter out certain kinds of content and give others priority service. But the universality of IP, and the many unexpected uses to which it has given birth, argue against such proposals to re-engineer the Internet. As information technology consultant Scott Bradner wrote, “We have the Internet that we have today because the Internet of yesterday did not focus on the today of yesterday. Instead, Internet technology developers and ISPs focused on flexibility, thus enabling whatever future was coming.”

Indeed, the entire social structure in which Internet protocols evolved prevented special interests from gaining too much power or building their pet features into the Internet infrastructure. Protocols were adopted by a working group called the Internet

THE FUTURE OF THE INTERNET— AND HOW TO STOP IT

This excellent book by Jonathan Zittrain (Yale University Press and Penguin UK, 2008) sees the vulnerabilities of the Internet—rapidly spreading viruses, and crippling attacks on major servers—as consequences of its essential openness, its capacity to support new inventions—what Zittrain calls its “generativity.” The book reflects on whether society will be driven to use a network of less-flexible “appliances” in the future to avoid the downsides of the Internet’s wonderfully creative malleability.

Engineering Task Force (IETF), which made its decisions by rough consensus, not by voting. The members met face to face and hummed to signify their approval, so the aggregate sense of the group would be public and individual opinions could be reasonably private—but no change, enhancement, or feature could be adopted by a narrow majority.

The larger lesson is the importance of minimalist, well-selected, open standards in the design of any system that is to be widely disseminated and is to stimulate creativity and unforeseen uses. Standards, although they are merely conventions, give rise to vast innovation, if they are well chosen, spare, and widely adopted.

Layers, Not Silos

Internet functionality could, in theory, have been provided in many other ways. Suppose, for example, that a company had set out just to deliver electronic mail to homes and offices. It could have brought in special wiring, both economical and perfect for the data rates needed to deliver email. It could have engineered special switches, perfect for routing email. And it could have built the ideal email software, optimized to work perfectly with the special switches and wires.

Another group might have set out to deliver movies. Movies require higher data rates, which might better be served by the use of different, specialized switches. An entirely separate network might have been developed for that. Another group might have conceived something like the Web, and have tried to convince ordinary people to install yet a third set of cables in their homes.

The magic of the hourglass structure is not just the flexibility provided by the neck of the bottle. It's the logical isolation of the upper layers from the lower. Inventive people working in the upper layers can rely on the guarantees provided by the clever people working at the lower layers, without knowing much about *how* those lower layers work. Instead of multiple, parallel vertical structures—self-contained silos—the right way to engineer information is in layers.

And yet we live in an information economy still trapped, legally and politically, in historical silos. There are special rules for telephones, cable services, and radio. The medium determines the rules. Look at the names of the main divisions of the Federal Communications Commission: Wireless, wireline, and so on. Yet the technologies have converged. Telephone calls go over the Internet, with all its variety of physical infrastructure. The bits that make up telephone calls are no different from the bits that make up movies.

Laws and regulations should respect layers, not the increasingly meaningless silos—a principle at the heart of the argument about broadcast regulation presented in Chapter 8.

End to End

“End to End,” in the Internet, means that the switches making up the core of the network should be dumb—optimized to carry out their single limited function of passing packets. Any functionality requiring more “thinking” than that should be the responsibility of the more powerful computers at the edge of the network. For example, Internet protocols could have been designed so that routers would try much harder to ensure that packets do not get dropped on any link. There could have been special codes for packets that got special, high-priority handling, like “Priority Mail” in the U.S. Postal Service. There could have been special codes for encrypting and decrypting packets at certain stages to provide secrecy, say when packets crossed national borders. There are a lot of things that routers might have done. But it was better, from an engineering standpoint, to have the core of the network do the minimum that would enable those more complex functions to be carried out at the edge. One main reason is that this makes it more likely that new applications can be added without having to change the core—any operations that are application-specific will be handled at the edges. This approach has been staggeringly successful, as illustrated by today’s amazing array of Internet applications that the original network designers never anticipated.

STUPID NETWORKS

Another way to understand the Internet’s end-to-end philosophy is to realize that if the computers are powerful at the edge of the network, the network itself can be “stupid,” just delivering packets where the packets themselves say they want to go. Contrast this with the old telephone network, in which the devices at the edge of the network were stupid telephones, so to provide good service, the switching equipment in the telephone office had to be intelligent, routing telephone signals to where the network said they should go.

Separate Content and Carrier

The closest thing to the Internet that existed in the nineteenth century was the telegraph. It was an important technology for only a few decades. It put

THE VICTORIAN INTERNET

That is the title of an excellent short book by Tom Standage (Berkley Books, 1999), making the argument that many of the social consequences of the Internet were seen during the growth of the telegraph. The content-carrier conflict is only one. On a less-serious level, the author notes that the telegraph, like the Internet, was used for playing games at a distance almost from the day it came into being.

an exclusive contract with Western Union, the telegraph monopoly. The contract gave the AP favorable pricing on the use of the wires. Other press services were priced out of the use of the “carrier.” And as a result, the AP got a lock on news distribution so strong that it threatened the functioning of the American democracy. It passed the news about politicians it liked and omitted mention of those it did not. Freedom of the press existed in theory, but not in practice, because the content industry controlled the carrier.

MORE ON INFORMATION FREEDOM

The SaveTheInternet.com Coalition is a pluralistic group dedicated to net neutrality and Internet freedom more generally. Its member organizations run the gamut from the Gun Owners of America, to MoveOn.org, to the Christian Coalition, to the Feminist Majority. Its web site includes a blog and a great many links. The blog of law professor Susan Crawford, scrawford.net/blog, comments on many aspects of digital information freedom, and also has a long list of links to other blogs.

the Pony Express out of business, and was all but put out of business itself by the telephone. And it didn’t get off to a fast start; at first, a service to deliver messages quickly didn’t seem all that valuable.

One of the first big users of the telegraph was the Associated Press—one of the original “wire services.” News is, of course, more valuable if it arrives quickly, so the telegraph was a valuable tool for the AP. Recognizing that, the AP realized that its competitive position, relative to other press services, would be enhanced to the extent it could keep the telegraph to itself. So it signed

Today’s version of this morality play is the debate over “net neutrality.” Providers of Internet backbone services would benefit from providing different pricing and different service guarantees to preferred customers. After all, they might argue, even the Postal Service recognizes the advantages of providing better service to customers who are willing to pay more. But what if a movie studio buys an ISP, and then gets creative with its pricing and service structure? You might discover that

your movie downloads are far cheaper to watch, or arrive at your home looking and sounding much better, if they happen to be the product of the parent content company.

Or what if a service provider decides it just doesn't like a particular customer, as Verizon decided about Nara? Or what if an ISP finds that its customer is taking advantage of its service deal in ways that the provider did not anticipate? Are there any protections for the customer?

In the Internet world, consider the clever but deceptive scheme implemented by Comcast in 2007. This ISP promised customers unlimited bandwidth, but then altered the packets it was handling to slow down certain data transmissions. It peeked at the packets and altered those that had been generated by certain higher-level protocols commonly (but not exclusively) used for downloading and uploading movies. The end-user computer receiving these altered packets did not realize they had been altered in transit, and obeyed the instruction they contained, inserted in transit by Comcast, to restart the transmission from scratch. The result was to make certain data services run very slowly, without informing the customers. In a net neutrality world, this could not happen; Comcast would be a packet delivery service, and not entitled to choose which packets it would deliver promptly or to alter the packets while handing them on.

In early 2008, AT&T announced that it was considering a more direct violation of net neutrality: examining packets flowing through its networks to filter out illegal movie and music downloads. It was as though the electric utility announced it might cut off the power to your DVD player if it sensed that you were playing a bootleg movie. A content provider suggested that AT&T intended to make its content business more profitable by using its carrier service to enforce copyright restrictions. In other words, the idea was perhaps that people would be more likely to buy movies from AT&T the content company if AT&T the carrier refused to deliver illegally obtained movies. Of course, any technology designed to detect bits illegally flowing into private residences could be adapted, by either governments or the carriers, for many other purposes. Once the carriers inspect the bits you are receiving into your home, these private businesses could use that power in other ways: to conduct surveillance, enforce laws, and impose their morality on their customers. Just imagine Federal Express opening your mail in transit and deciding for itself which letters and parcels you should receive!

Clean Interfaces

The electric plug is the interface between an electric device and the power grid. Such standardized interfaces promote invention and efficiency. In the

Internet world, the interfaces are the connections between the protocol layers—for example, what TCP expects of IP when it passes packets into the core, and what IP promises to do for TCP.

In designing information systems, there is always a temptation to make the interface a little more complicated in order to achieve some special functionality—typically, a faster data rate for certain purposes. Experience has shown repeatedly, however, that computer programming is hard, and the gains in speed from more complex interfaces are not worth the cost in longer development and debugging time. And Moore’s Law is always on the side of simplicity anyway: Just wait, and a cruder design will become as fast as the more complicated one might have been.

Even more important is that the interfaces be widely accepted standards. Internet standards are adopted through a remarkable process of consensus-building, nonhierarchical in the extreme. The standards themselves are referred to as RFCs, “Requests for Comment.” Someone posts a proposal, and

RFCs AND STANDARDS

The archive of RFCs creates a history of the Internet. It is open for all to see—just use your favorite search engine. All the Internet standards are RFCs, although not all RFCs are standards. Indeed, in a whimsically reflexive explanation, “Not all RFCs are standards” is RFC 1796.

a cycle of comment and revision, of buy-in and objection, eventually converges on something useful, if not universally regarded as perfect. All the players know they have more to gain by accepting the standard and engineering their products and services to meet it than by trying to act alone. The Internet is an object lesson in creative compromise producing competitive energy.

[Curriculum \(/bjc-course/curriculum\)](#) / [Unit 10 \(/bjc-course/curriculum/10-data\)](#) /

[Reading 1 \(/bjc-course/curriculum/10-data/readings/01-how-google-works\)](#) /

How Google Works

The Problem of Scale

One of the hardest parts in the life of a search engine is managing the extraordinary diversity of content it encounters while doing its job (both in the pages it sees and the questions that it gets asked). Instead of trying to do all of the reasoning and analysis when a search term arrives, it tries instead to organize the information in such a way that it learns a little bit about it, and where to find it, before its even asked a question. If search engines were actually going out to every page to look at its content for the first time when you performed a search, it would take forever to get your results. Imagine going to a big concert and being asked to find the oldest person from Indiana at the show. This would be incredibly difficult if you didn't have any information about the people – but if you were allowed to arrange the people by the origin state as well as their age within each state before the question was asked, then it would become much easier to arrive at a quick and accurate solution. Phone books are alphabetical for a reason!

The Anatomy of a Search Engine

Search engines, including Google, are generally composed of three major parts: the crawler, the indexer, and the searcher. This video was produced by a team at Google describing how search engines work; listen to their description of the process (they are pretty good at it, after all). Don't worry if you don't follow all of it – we'll spend more time on it in a second.

Sounds easy enough, right? Today we're going to use BYOB to create a very simple search engine. Search is a reasonably challenging thing to do at all, and an incredibly difficult thing to master (Google, Microsoft, and some others are spending billions trying!). Breaking up the search engine into multiple parts will make the overall problem much more manageable. Understanding the concepts behind search can be challenging, so we'll spend a little bit more time on those in the following section.

Curriculum (/bjc-course/curriculum) / Unit 10 (/bjc-course/curriculum/10-data) /
Reading 2 (/bjc-course/curriculum/10-data/readings/02-life-before-search) /

Life Before Search

Now consider a single cluster of technologies: the Internet and web search. These technologies have only been popular for 20 - 30 years, but it's hard to imagine life before them.

Before Search Engines and the Internet

Before the Internet became the phenomenon that it is today, it was considerably more difficult to share information with others, especially if they weren't in the same area of the world. In addition to you and your classmates' submissions in the discussion forum, CyberNetNews (<http://cybernetnews.com/cybernotes-life-before-the-internet-was-like/>) mentions a number of differences between then (15 years ago) and now.

The Internet today provides a tremendous service of making information available to anyone in the world that can connect to it. "Information" can take the form of company websites, email, instant messaging, audio / video, and many, many other possibilities.

Early Search Engines

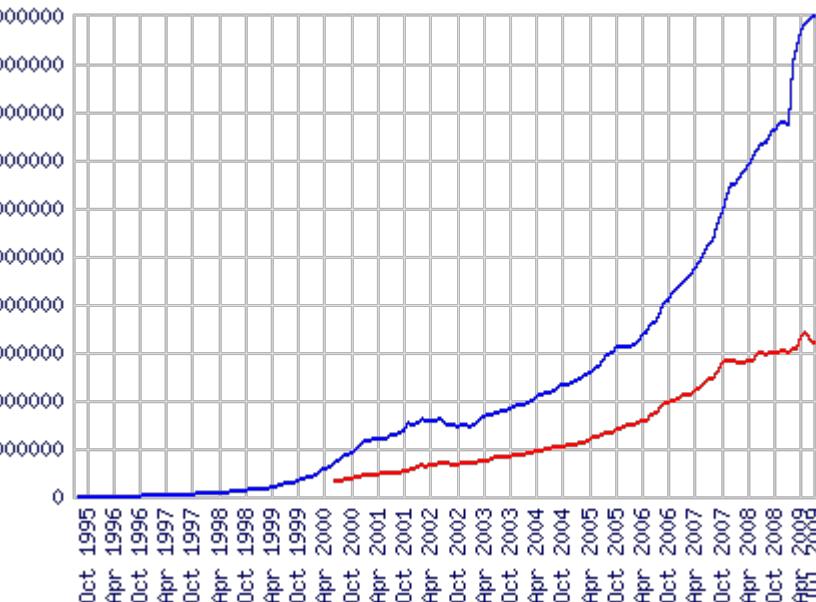
The time of the Internet before search engines was tough as well. The Internet was the same decentralized, wildly diverse collection of information that it is today (although a good bit smaller), but was also terribly difficult to navigate: it was comparable to a telephone network without a phone book. Finding your way to new places on the Internet often involved someone telling you about their site, or stumbling across a site from one you were already familiar with.

Not being ones to shy from problems, computer scientists noticed this shortcoming and began working on technology that would automatically crawl and index the websites that they could find. The directories they created became early search engines. A number of search engines like Lycos, Dogpile, Altavista and Askjeeves used to be big players in the search market. These search engines made huge steps in organizing the Internet by bringing most of it to one place. Nevertheless, the search process was unrefined by modern standards and often required users to search through long lists of sites to find what they needed.

Alas, this wasn't the case for long. Tremendous progress has been made between then and now, much of which has come from a single company that devoted its entire business to search: Google.

Today's Search

Almost anyone these days is able to find information from across the world in less than half of a second. You can answer almost any question whose solution is known by humanity in under 30 seconds. The amount of information that is available online has taken the world by storm, shifting entire industries and antiquating others. There are tens of millions of websites online at any one time and more are being added every year.



If there is so much information available on the Internet, it seems pretty remarkable that today's search engines can come up with such startlingly accurate results for what we want, when we want it.

Search Takes a Team

The inner workings of search engines are pretty complex systems and can be very intimidating. But what if we were to look at search engines differently? What if we saw them instead as a fruit stand?



There may have been a time when this fruit stand could've been run by one person, but now the demand at the stand is far too high for one person and the job is divided up into three different tasks:

Crawler

The task of the Crawler is straightforward: search all of the fields and forests that you can find for fruit. It doesn't matter what kind it is, what it tastes like, or whether you think that anyone will want to eat it. You find it, you take it. The performance of a Crawler could be measured by how much fruit it could gather over time – the faster the better. Big fruit stands may have multiple crawlers that can easily do their jobs concurrently.

Indexer

The Indexer is the heart of the stand's business. After the Crawler(s) delivers its harvest, the Indexer goes to work organizing the fruit in a way that will be convenient for any Searchers that happen to come by. Depending on the kind of Searchers that the stand typically sees, the Indexer may organize the fruit by type, age, color, size, or otherwise. The cart itself can then be called the index, which is the place where the Indexer places all of the

organized fruit. Notice that the word “index” has been used before in this course (in the context of lists as well as loops)...this is the same word but with a different meaning! The Indexer puts the fruit on the stand in an order that it can remember so that it can quickly find a particular type of fruit when a Searcher asks for it.

Searcher

the Searcher is the customer who is looking for the perfect fruit. Different Searchers are usually looking for different fruit, and can often be very picky with their requirements for each individual piece. They search through the fruit stand and find the fruit that best fits what they’re looking for, and want to get in, find their fruit, and get out as quickly as possible.

Curriculum (/bjc-course/curriculum) / Unit 10 (/bjc-course/curriculum/10-data) /
Reading 4 (/bjc-course/curriculum/10-data/readings/04-bits-questions) /

Reading Assignment

Read the Appendix of Blown to Bits (15 pages). This chapter talks about the Internet as a system.

Reading Questions

Post answers to the following questions on a Portfolio page. We will discuss these in class.

Explain each of the following terms

- router
- ISP
- IP Address
- Domain name
- IP Protocol
- TCP Protocol
- HTTP Protocol
- SMTP Protocol

Answer the following questions:

- What's the difference between a circuit switched and a packet switched network?
 - How have open standards helped the Internet develop?
 - What is net neutrality and why is it important?
-

Lab: How the Internet Works

Key Concept A. The Internet is a network of autonomous systems.

Learning Objective 24: The student can explain the abstractions in the Internet and how the Internet functions.

- 24a. Explanation of how the Internet connects devices and networks all over the world.
- 24b. Explanation of how the Internet and the systems built on it facilitate collaboration.
- 24c. Description of evolving standards that the Internet is built on, including those for addresses and names.
- 24d. Identification of abstractions in the Internet and how the Internet functions.

Let's Watch the Internet

Cnet provides a tool that measures the bandwidth of your Internet connection. Bandwidth (http://en.wikipedia.org/wiki/Bandwidth_%28computing%29) is the amount of data that can be transferred over your connection per unit time. It is usually measured in kilobits per second (Kbps), thousands of bits per second, or Millibits per second (Mbps), millions of bits per second.

Akamai (http://en.wikipedia.org/wiki/Akamai_Technologies) provides some nice visualization tools (http://www.akamai.com/html/technology/visualizing_akamai.html).

There are a couple of command-line tools that we can use to watch packets traveling on the Internet. But first you have to find the command line on your system:

- Find the Terminal or Konsole program on your workstations.
- (Linux Users: Look under System)
- (Mac Users: Open Applications < Utilities < Terminal)
- (Windows User: Open the Command Prompt)

Traceroute (<http://en.wikipedia.org/wiki/Traceroute>) is a network diagnostic tool that lets you display the path that your packets take across the Internet.

```
$ traceroute google.com
```

```
traceroute: Warning: google.com has multiple addresses; using 74.125.224.163
```

```
traceroute to google.com (74.125.224.163), 64 hops max, 52 byte packets
```

```
1 192.168.1.1 (192.168.1.1) 1.302 ms 0.919 ms 0.757 ms
```

```
2 10.4.144.1 (10.4.144.1) 8.513 ms 9.206 ms 8.140 ms
```

```
3 68.9.8.225 (68.9.8.225) 10.512 ms 10.055 ms 9.773 ms
```

```
4 ip68-9-7-65.ri.ri.cox.net (68.9.7.65) 11.274 ms 10.933 ms 11.802 ms
```

```
5 ip98-190-33-42.ri.ri.cox.net (98.190.33.42) 9.947 ms 9.830 ms 10.313 ms
```

```
6 ip98-190-33-21.ri.ri.cox.net (98.190.33.21) 13.648 ms 12.782 ms 11.729 ms
```

```
7 provdsrj01-ae3.0.rd.ri.cox.net (98.190.33.20) 11.678 ms 12.195 ms 11.936 ms
```

```
8 nyrbprj01-ae2.0.rd.ny.cox.net (68.1.1.173) 17.775 ms 17.548 ms 17.816 ms
```

```
9 209.85.248.178 (209.85.248.178) 21.314 ms 17.731 ms 22.360 ms
```

```
10 209.85.251.88 (209.85.251.88) 18.121 ms 21.036 ms 18.488 ms  
11 209.85.249.11 (209.85.249.11) 28.026 ms 55.017 ms 28.288 ms  
12 66.249.95.149 (66.249.95.149) 39.955 ms 38.041 ms 50.290 ms  
13 72.14.233.86 (72.14.233.86) 56.257 ms 65.504 ms 54.386 ms  
14 64.233.174.142 (64.233.174.142) 98.997 ms 91.343 ms 89.584 ms  
15 64.233.174.189 (64.233.174.189) 90.320 ms 89.137 ms 168.799 ms  
16 72.14.236.11 (72.14.236.11) 90.281 ms 92.398 ms 100.019 ms  
17 lax02s01-in-f3.1e100.net (74.125.224.163) 91.074 ms 99.050 ms 92.037 ms
```

\$

Ping (<http://en.wikipedia.org/wiki/Ping>) is a utility to test whether a host on the Internet is reachable: that lets you display the path that your packets take across the Internet and reports the round-trip time for messages to that host.

```
$ ping google.com
```

```
PING google.com (74.125.239.14): 56 data bytes
```

```
64 bytes from 74.125.239.14: icmp_seq=0 ttl=48 time=91.242 ms  
64 bytes from 74.125.239.14: icmp_seq=1 ttl=48 time=89.957 ms  
64 bytes from 74.125.239.14: icmp_seq=2 ttl=48 time=91.471 ms  
64 bytes from 74.125.239.14: icmp_seq=3 ttl=48 time=90.506 ms  
64 bytes from 74.125.239.14: icmp_seq=4 ttl=48 time=91.070 ms
```

...

Internet Infrastructure

The Internet is a network of independent networks – independent in the sense that the local networks use different protocols (<http://sumansinformationtechnology.blogspot.com/2010/01/what-is-internet.html>) to transmit data among their computers.

Routers running the Internet Protocol (IP) (http://en.wikipedia.org/wiki/Internet_Protocol) connect the different local networks together, creating the Internet. The IP takes care of routing data through the Internet and translates data from a local protocol (such as Ethernet) to IP and vice versa.

Analogy: You can think of the local networks as different countries where the citizens are connected by different languages. The IP is like a translator that translates from French to English.

Example

Suppose you type the URL of your school's home page into your browser. It may seem like your browser (the client) is directly connected to my web page (on the server), as in the top half of the following diagram (http://en.wikipedia.org/wiki/File:IP_stack_connections.svg).

But your request and the server's response travel through several Internet abstraction layers as shown in the bottom half of this diagram.

Tracing the Data Flow

At the **application layer**:

- Your browser uses the HyperText Transfer Protocol (HTTP) (http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) protocol, the primary World Wide Web (WWW) protocol.
- The HTTP protocol requests a translation of the host name portion of the URL www.cs.trincol.edu into an IP Address (<http://turing.cs.trincol.edu/~ram/cpsc110/inclass/internet/>) of the server that stores my home page from a Domain Name System (DNS) (http://en.wikipedia.org/wiki/Domain_Name_System) server.

There are many other application layer protocols, including:

- File Transfer Protocol (FTP) (http://en.wikipedia.org/wiki/File_Transfer_Protocol)
- Simple Mail Transfer Protocol (SMTP) (http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol)
- Post Office Protocol (http://en.wikipedia.org/wiki/Post_Office_Protocol)
- Secure Shell (http://en.wikipedia.org/wiki/Secure_Shell) – remote terminal sessions

At the **transport layer**:

- The Transmission Control Protocol (TCP) (http://en.wikipedia.org/wiki/Transmission_Control_Protocol) is used to insure a reliable, ordered delivery of a stream of data from your browser to the trincol.edu server.

At the **internet layer**:

- The Internet Protocol (IP) (<http://turing.cs.trincol.edu/~ram/cpsc110/inclass/internet/>) is used to transmit 1500 byte packets of data through the internet. Each packet is sent separately from the client to the server.

At the link layer, the **hardware layer**.

Various protocols are used to transmit the data across different local area networks. Some of the protocols include:

- Ethernet (http://en.wikipedia.org/wiki/Transmission_Control_Protocol)
- Digital Subscriber Line (DSL) (http://en.wikipedia.org/wiki/Digital_subscriber_line)
- Point-to-Point (PPP) (http://en.wikipedia.org/wiki/Point-to-Point_Protocol)

Packet Switching

The Internet (the IP) is based on packet switching (http://en.wikipedia.org/wiki/Packet_Switching). Data are broken into 1500 byte blocks (8 bits per byte), which are transmitted from router to router through the Internet.

This contrasts with **circuit switching**, the technology that land-line telephones used to use, in which a continuous circuit was set up through switches from one end of the call to the other.

IP Addresses

An IPv4 address (http://en.wikipedia.org/wiki/IP_address) uses a 32-bit IP address, broken into 4 8-bit segments each represented by a decimal number. An IPv6 address uses a 128-bit address, broken into 8 16-bit segments represented as Hexadecimal numbers:

Question: Why do you think Hex is used in IPv6 instead of decimal?

Domain Names

An Internet domain name is organized into a number of levels as shown in this diagram (http://en.wikipedia.org/wiki/Domain_name). A domain name takes the following form:

fourth-level-domain . third-level-domain . second-level-domain . top-level domain

turing . cs . trincoll . edu

A hostname is a domain name that is associated with an IP address. For example, the following are examples of hostnames:

- trincoll.edu
- www.trincoll.edu
- turing.cs.trincoll.edu

But top-level domain names, such as com and edu are not hostnames.

Domain Name Service

Domain names are mapped into IP address by the Domain Name System (http://en.wikipedia.org/wiki/Domain_Name_System), a network of servers that keep track of the mappings.

In this example, three look-ups are need before the IP address is resolved.

The World Wide Web

The World Wide Web (WWW) is not a network – technically speaking. It is a system of interlinked hypertext documents (<http://turing.cs.trincoll.edu/~ram/cpsc110/inclass/internet/>) that can be accessed via the Internet.

The Web is a service governed by the HyperText Transfer Protocol (HTTP) (http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) protocol.

It was invented by Sir Tim Berners-Lee (http://en.wikipedia.org/wiki/Tim_Berners-Lee), who, to his credit, turned his invention into an open standard.

“I just had to take the hypertext idea and connect it to the Transmission Control Protocol and domain name system ideas and ta-da! the World Wide Web.”

Discussion Questions

- Would the Web be the success that it is if Berners-Lee had made it a proprietary system?
- Would the Internet be the success that it is if it was not based on open standards?

Curriculum (/bjc-course/curriculum) / Unit 10 (/bjc-course/curriculum/10-internet) / Lab 2 (/bjc-course/curriculum/10-internet/labs/02-internet-activity) /

Internet Activity

This activity is a followup of How the Internet works. Create a single written document (such as Microsoft Word) that answers the questions below. Work by yourself or with a partner to answer these questions.

The purpose is to try to understand something about what the Internet is and explore a few examples of how you can use computation and computational thinking in ways related to the Internet.

There are four general parts to this: a map of the internet, understanding that the Internet is a network of autonomous systems (regional networks), thinking about IP addresses and DNS — the Domain Name System, and tracking how packets travel on the Internet.

Each of the following four sections have questions that you (and your partner, if you have one) will answer.

Map of the Internet (10-15 minutes)

First let's look at a few examples to get the conversation started. Here's a website that says it's a map of the Internet: <http://internet-map.net/> (<http://internet-map.net/>)

Click around, zoom, and in general play with this for a couple of minutes.

Task: Add a question or observation you have about the map. There can be more than one.

There are other maps of the Internet.

<http://www.peer1.com/map-of-the-internet> (<http://www.peer1.com/map-of-the-internet>)

<http://www.telegeography.com/telecom-maps/global-internet-map/> (<http://www.telegeography.com/telecom-maps/global-internet-map/>)

<http://xkcd.com/195/> (<http://xkcd.com/195/>)

Autonomous Systems (10-15 minutes)

Conventionally the Internet is often referred to as a “network of networks”. A slightly different view is that the Internet is a network of autonomous systems that communicate with each other using the Border Gateway Protocol (BGP). Let's find out something about autonomous systems in this activity.

Task: Answer the three questions about Autonomous Systems

- How many autonomous systems (AS) are there today on the Internet?
- What is the AS number of Duke University?
- What entity/company is associated with AS 16631?

IP Addresses and Domain Names (10-15 minutes)

An IP address identifies a machine/device connected to the Internet; it's a 32- or 128-bit number, e.g., 184.84.228.110 for an IPv4 dotted-quad. People like names better than machines, names like whitehouse.gov. We'll explore names and addresses, specifically how they tie together and how they work (briefly).

You can use <http://www.kloth.net/services/nslookup.php> (<http://www.kloth.net/services/nslookup.php>) or <http://www.kloth.net/services/dig.php> (<http://www.kloth.net/services/dig.php>) to answer these questions, there are other services as well that wrap dig and nslookup in a website. For geolocation of an IP address you can try <http://geomaplookup.net/> (<http://geomaplookup.net/>) or <http://www.infosniper.net/> (<http://www.infosniper.net/>)

Task: Enter answers to the following four questions

- What is the domain name registered to 65.39.205.54?
- What is the IP address associated with girlswocode.com?
- What is the IP address associated with csprinciples.org?
- In what city is the IP address of csprinciples.org located?

Traceroute (10-15 minutes)

The traceroute command shows the “hops” that a packet will take in getting from one place on the Internet to another. This command/tool can be run via several websites, some that offer visualizations. We’ll use <http://traceroute.org> (<http://traceroute.org>) which maps several sites in other countries to your IP address or one you specify. For visualizing you can try either <http://traceroute.monitis.com> (<http://traceroute.monitis.com>) or <http://www.yougetsignal.com/tools/visual-tracert/> (<http://www.yougetsignal.com/tools/visual-tracert/>)

For example, using the pages at traceroute.org, if you start from the site given in the country of Kyrgyzstan 21 hops are shown to reach Duke (output from website)

traceroute to 152.3.250.1 (152.3.250.1), 30 hops max, 72 byte packets

```
1 r2d2 (212.42.96.42) 0.255 ms 0.262 ms 0.246 ms (0% loss)
2 R76 (212.42.96.80) 0.340 ms 0.310 ms 0.314 ms (0% loss)
3 195.218.197.85 (195.218.197.85) 57.587 ms 57.609 ms 74.506 ms (0% loss)
4 p4.Moscow.gldn.net (195.239.11.166) 81.353 ms 85.248 ms 83.839 ms (0% loss)
5 mx01.Stockholm.gldn.net (194.186.156.41) 75.101 ms 75.684 ms 75.102 ms (0% loss)
6 xe-11-0-0-xcr1.skt.cw.net (166.63.220.65) 75.182 ms 90.087 ms 75.216 ms (0% loss)
7 xe-0-3-1-xcr2.amd.cw.net (195.2.9.217) 201.529 ms 176.642 ms 177.156 ms (0% loss)
8 ae0-xcr2.amd.cw.net (195.2.30.105) 175.810 ms 175.820 ms 175.656 ms (0% loss)
9 xe-7-0-0-xcr2.lsw.cw.net (195.2.25.38) 176.593 ms 176.578 ms 176.607 ms (0% loss)
10 xe-2-2-0-xcr2.lnd.cw.net (195.2.28.185) 178.416 ms 175.892 ms 175.479 ms (0% loss)
11 xe-3-1-0-xcr1.bkl.cw.net (195.2.21.214) 103.025 ms 102.920 ms 103.093 ms (0% loss)
12 xe-8-1-0-xcr1.nyk.cw.net (195.2.25.26) 176.424 ms 176.293 ms 200.133 ms (0% loss)
13 nyc-brdr-02.inet.qwest.net (63.146.27.61) 174.859 ms 174.929 ms 174.798 ms (0% loss)
14 * * * (100% loss)
15 65.121.156.210 (65.121.156.210) 200.693 ms 216.372 ms 200.503 ms (0% loss)
16 rpcrs-gw-to-wscrs-gw.ncren.net (128.109.212.9) 223.205 ms 203.419 ms 203.567 ms (0% loss)
17 rlgh7600-gw-to-rtp-crs-gw.ncren.net (128.109.9.6) 206.194 ms 206.184 ms 206.192 ms (0% loss)
18 roti-gw-to-rlgh7600-gw.ncren.net (128.109.70.18) 215.218 ms 215.790 ms 215.372 ms (0% loss)
19 tel1u-datacenter-vrf.netcom.duke.edu (152.3.234.25) 204.385 ms 204.414 ms 205.012 ms (0% loss)
```

20 tel2u_3_12_4.netcom.duke.edu (152.3.250.254) 215.001 ms 215.041 ms 214.813 ms (0% loss)

21 dukedns1.netcom.duke.edu (152.3.250.1) 204.437 ms 204.365 ms 204.392 ms (0% loss)

Your task is to pick 3 cities/locations from <http://traceroute.org> (<http://traceroute.org>) and use tools to find the number of hops to either your machine (sometimes that's the default) or to Duke's nameserver which is 152.3.250.1 — if you have a choice of how to connect, use traceroute. Many of the links at traceroute.org don't work, click around as needed.

Task: Enter the three cities and the number of hops

Reflection on these Internet Tools

- What impact has the Internet had on solving problems and gathering information?
 - What problems has the Internet created? What are we doing to address these problems?
-

This is a test/project grade.

Portfolio Project: Internet Learning Objectives

The Internet Portfolio Task addresses the following CS Principles Learning Objectives:

- 27: The student can explain the abstractions in the Internet and how the Internet functions.
- 28: The student can explain characteristics of the Internet and the systems built on it.
- 29: The student can analyze how characteristics of the Internet and systems built on it influence their use.
- 30: The student can connect the concern of cybersecurity with the Internet and systems built on it.
- 31: The student can analyze how computing affects communication, interaction, and cognition.
- 33: The student can connect computing with innovations in other fields.
- 34: The student can analyze the beneficial and harmful effects of computing.
- 35: The student can connect computing within economic, social, and cultural contexts.

Task

- Identify and describe a significant, contemporary problem and potential solution that are connected by the Internet to a societal, economic, or cultural context. The problem and/or potential solution must have a strong connection to computing and the Internet.
- You will create a document describing the problem, the potential solution, the societal, economic, or cultural contexts, cyber security concerns, and the relevant characteristics of the Internet. The societal, economic, or cultural context should affect a significant population more than a few hundred people).
- Note: In the task description and the text below the phrase “problem and/or solution” means that either the problem, or the solution, or both should have the characteristics referred to. For example, the task description requires that the problem, or the solution, or both, must have a strong connection to computing and the Internet.

Prepare and submit the following

A single written document that addresses the task and meets all the requirements listed below.

- The document can include illustrations and non&textual examples.
- The document does not need to be an essay. For example, it can be a sequence of written paragraphs each of which addresses one of the requirements below, but which does not include words to connect the paragraphs together.
- You can also write an essay in which the paragraphs are connected.

Requirements

- Maximum length of essay is 1000 words.
- MLA Format, Double-spaced, Times New Roman 11 pt
- Works Cited, MLA format
- You will also do a 30-second Talking Point in class
- Identify at least one cyber security concern related to the problem and/or the potential solution depending on which of these are connected to the Internet).

- Describe the cyber security concern and explain how it relates to the problem or its potential solution or both.
- Provide a brief but clear description of the problem.
- Identify and describe the problem clearly.
- What population is affected and how?
- If the problem is connected to the Internet describe how. Be sure to address the specific characteristics of the Internet (see Description of Internet Characteristics) relevant to this problem and how the Internet and the problem are connected.
- Provide a brief but clear description of the potential solution.
- Describe the potential solution clearly and how it addresses the problem.
- If the solution is connected to the Internet describe how. Be sure to address the specific characteristics of the Internet (see Description of Internet Characteristics) relevant to this solution and how they enable the solution.
- Connect your references to the descriptions of the problem and the solution. That is, your descriptions must make explicit references to your sources.
- Describe one potentially beneficial and one potentially harmful effect the Internet has on society, culture or economics in the context of the problem or the potential solution.
- Identify and describe one field other than computing that contributed to or is impacted by the problem or the proposed solution, citing at least one specific example. Describe how the field contributes to or is impacted by the problem or its solution.

Your document must make references to two external sources that provide context for the problem and solution.

- The references should be high quality sources.
- Each reference should include a URL when appropriate, but also a citation for the reference, e.g., the author and title of the reference.
- You must include a date on which you accessed the source if the source is online.
- Each reference should be to something written between September 2012 and today.
- The references should be to sources of information that anyone can use to learn about the problem or its solution.

Description of Internet Characteristics

When you describe the characteristics of the Internet relevant to the problem or its solution or both) you must identify, describe, and explain at least the characteristics below that are relevant and related to the problem or the solution. You only need to identify, describe, and explain those characteristics below that are relevant and related to the problem and/or solution. You may identify characteristics of the Internet in addition to those below.

- Evolving Internet standards and abstractions e.g., addresses and names)
- Hierarchy and redundancy in the Internet
- Interfaces and protocols of the Internet that enable widespread use
- The trust model of the Internet and its role in cybersecurity
- How cryptography affects cybersecurity

Rubric

Essay (85 pts)

- Identify and describe a significant, contemporary problem
- Identify and describe a potential solution to the problem
- Identify societal, economic, or cultural context of the problem/solution
- Describe how population is impacted
- Identify and describe the problem's/solution's connection to the Internet
- Identify & describe at least one cyber security concern related to the problem and/or the potential solution

- Identify and describe one field other than computing that contributed to or is impacted by the problem or the proposed solution, citing at least one specific example. Describe how the field contributes to or is impacted by the problem or its solution.
- At least two appropriate references used as assigned

Works Cited (10 pts)

30 sec Talking Points (5 pts)

Unit 11: Game Development

Games with a Purpose

Learning Objectives

Readings/Lectures

- Lecture Slides 11.01: Video Games ([/bjc-course/curriculum/11-game-development/readings/01-video-games-slides.pdf](#))
- Reading 11.02: Fundamentals of Game Design ([/bjc-course/curriculum/11-game-development/readings/02-fundamentals-of-game-design](#))
- Reading 11.03: Game Design Principles ([/bjc-course/curriculum/11-game-development/readings/03-game-design-principles](#))

Labs/Exercises

- Portfolio Project 11.01: Programming Portfolio Project ([/bjc-course/curriculum/11-game-development/labs/03-internet-portfolio-project](#))
-

VIDEO GAMES

The Beauty and Joy of Computing/CS Principles



UC Berkeley EECS
Lecturer SOE
Dan Garcia



UC Berkeley EECS
TA-in-Training
Glenn Sugden

GAMIFICATION OF BUSINESS!

Channeling the “gamer addiction” to earn virtual points, companies are now adding badges and rewards to things.

E.g., Nike + (exercise game), Mint.com (encouraging savings), Foursquare (location-based social network), etc...



tech.fortune.com/2010/09/03/the-game-based-economy/

How big is US video game market?

- a) \$100,000,000
- b) \$1,000,000,000
- c) \$10,000,000,000
- d) \$100,000,000,000
- e) \$1,000,000,000,000



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (3)

Garcia, Fall 2010

How big is US video game market?

- a) \$100,000,000
- b) \$1,000,000,000
- c) \$10,000,000,000
- d) \$100,000,000,000
- e) \$1,000,000,000,000



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (4)

Garcia, Fall 2010

Video Games : Overview

- History
 - Inventors & Games
- How
 - Design
 - 2D & 3D graphics
 - Motion Capture
 - Artificial Intelligence (AI)
- Good, Bad, Ugly
 - GWAP, RSI, Violence
- Future



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (5)

Garcia, Fall 2010

Documentaries on Video Games

- History: Video Games: Behind the Fun (2000)
 - Available on Netflix
- PBS: The Video Game Revolution (2004)
 - video.google.com/videoplay?docid=-4729348985218842392
- Discovery: History of Video Games (2006)
 - video.google.com/videoplay?docid=3637639460474263178
- ON Networks : Play Value (2009)
 - www.onnetworks.com/videos/play-value
- History of Video Games (WWW)
 - en.wikipedia.org/wiki/History_of_video_games
 - en.wikipedia.org/wiki/List_of_films_based_on_video_games#Documentaries_on_video_games



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (6)

The Beginning : Spacewar!

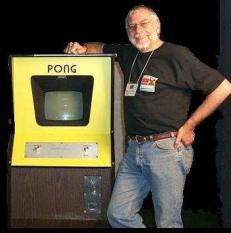
- First to gain recognition
 - Others had games before
 - “Conceived in 1961 by Martin Graetz, **Stephen Russell**, & Wayne Wiitanen”
 - Written for PDP-1 @ MIT
 - Inspired lots, widely ported
- Can still play this!
 - 1 Working PDP-1 ... in CHM
 - Java version available



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (7)

The Founding Fathers

- Ralph Baer
- Nolan Bushnell



www.onnetworks.com/videos/play-value/the-founding-fathers
(also on iTunes in HD 720p)



Garcia, Fall 2010 CC BY NC SA

Shigeru Miyamoto

- The “Walt Disney” of computing gaming
 - Chief Game designer at Nintendo
 - 1st elected to Hall of Fame
- Designed (among others):
 - Donkey Kong
 - Super Mario Bros
 - The Legend of Zelda
 - Super Mario 64
 - Nintendo DS, Wii



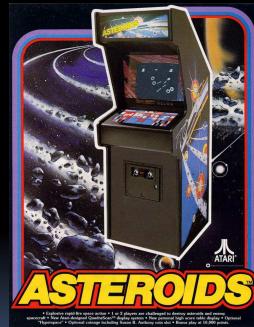
www.onnetworks.com/videos/play-value/shigeru-miyamoto
www.time.com/time/asia/2006/heroes/bl_miyamoto.html
en.wikipedia.org/wiki/Shigeru_Miyamoto

Garcia, Fall 2010 CC BY NC SA

UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (9)

History of Video Games : 1970s

- Golden age of video arcades
 - Pong, Space Invaders, Asteroids, Pac Man
- 1st gen consoles (1972–1976)
 - Magnavox Odyssey
- Mainframe computers
 - Hunt the Wumpus, Rogue
- Home computers
 - Type the program in!
 - Floppies, Tapes, Zork, others.
- 2nd gen consoles (1977–1984)
 - Atari 2600, Intellivision, ColecoVision, Activision



en.wikipedia.org/wiki/History_of_video_games
www.thegameconsole.com
Garcia, Fall 2010 CC BY NC SA

History of Video Games : 1980s

- Genre innovation
- Gaming computers
 - Apple II, Commodore 64, Atari 800
- Early online gaming
 - Mostly text only, MUDs
- Handheld LCD games
- Video game crash of 1983
 - Atari buried millions of ETs in dump
- 3rd gen consoles (1985–1989)
 - Nintendo Ent. System (NES)
 - Super Mario Bros, Zelda, FF I
 - Gamepad introduced



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (11)

History of Video Games : 1990s

- Decline of arcades
- Handhelds come of age
 - GameBoy, Sega Game Gear
- Mobile phone gaming
- Fourth generation consoles (1990–1994)
 - Sega Genesis, Super NES
- Fifth generation consoles (1995–2000)
 - Playstation, Nintendo 64 (with Super Mario 64)
- Transition to 3D, CDs
 - Crash Bandicoot, Tomb Raider



UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (12)



Garcia, Fall 2010 CC BY NC SA

History of Video Games : 2000s

- Mobile games
 - iPhone (games ½ apps)
- Sixth generation consoles (since 2001)
 - PS2, Xbox, GameCube
 - Return of alternate controllers (DDR, guitars)
- Online gaming rises to prominence
 - WoW, Ultima Online
- Rise of casual PC games
 - Bejeweled, The Sims



UC Berkeley CS10 "The Beauty and Joy of Computing" : Video Games (13)

Garcia, Fall 2010
CC BY NC SA

History of Video Games : 2005+

- Seventh generation consoles (since 2005)
 - Portables
 - Nintendo DS, PSP, iPhone
 - Consoles
 - PS3, Xbox 360, Wii
- Increases in development budgets
- Motion control revolutionizes play
 - Wii controller, iPhone



UC Berkeley CS10 "The Beauty and Joy of Computing" : Video Games (14)

Garcia, Fall 2010
CC BY NC SA

Example: Playstation 3 Hardware

- State-of-the-art system
 - But SW determines success!
 - (also, cool controllers helps)
- 9 3.2GHz Cores (1PPE, 8SPE)
 - Power Processing Elt (PPE)
 - Supervises activities, allocates work
 - Synergistic Processing Elt (SPE)
 - Where work gets done
 - During testing, one "locked out"
 - I.e., it didn't work; shut down



en.wikipedia.org/wiki/PlayStation_3
www.us.playstation.com

Garcia, Fall 2010
CC BY NC SA

UC Berkeley CS10 "The Beauty and Joy of Computing" : Video Games (15)

Design of a Casual Video Game

- Staff requirements
 - Can be done by one person, ala days of old
 - Bigger teams also (< 10)
 - Lots of new developers
- Phones great platforms
 - iPhone dominates field
 - Students are signing up!
- Time to completion
 - Often only a few months!

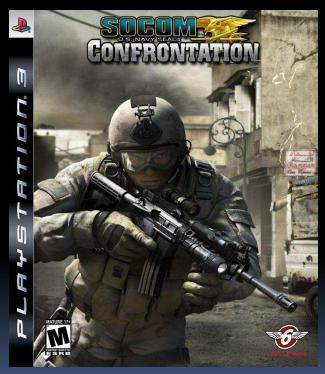


www.apple.com/iphone/apps-for-everything/fun-and-games.html
blog.entrepreneur.com/2009/07/7-addicting-casual-games
en.wikipedia.org/wiki/Casual_game

Garcia, Fall 2010
CC BY NC SA

Design of a Core Video Game

- Staff requirements
 - Cross-disciplinary
 - Producer, programmers, game, graphic & sound designers, musicians, testers, ...
 - 100+ person teams
- Similar to film
 - Often, games->film, and film->games
 - Lucasfilm, etc. want to tie assets together



en.wikipedia.org/wiki/Video_games
Garcia, Fall 2010

CC BY NC SA

UC Berkeley CS10 "The Beauty and Joy of Computing" : Video Games (17)

% of Parents "Games positive for kids"



- 34%
- 44%
- 54%
- 64%
- 74%



entertainment software association

Garcia, Fall 2010
CC BY NC SA

UC Berkeley CS10 "The Beauty and Joy of Computing" : Video Games (18)

www.theesa.com/facts

% of Parents “Video games positive for kids”




a) 34%
b) 44%
c) 54%
d) 64%
e) 74%

Garcia, Fall 2010
 UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (19)
 CC BY NC SA

How : 3D Computer Graphics

- Similar to making a 3D animated film...
 - Model characters, environment in 3D
 - Add shading + lights + effects + behavior
 - Let 3D rendering engine (on graphics card) do the work of figuring out 2D scene from 3D
- Limitations
 - Many things are too “expensive” to do in 30 frames per second
 - Research breakthroughs!



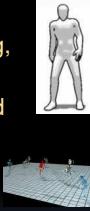
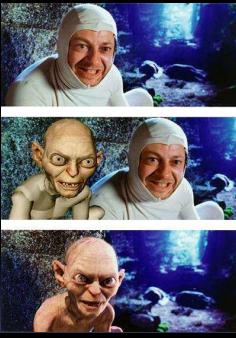
www.nytimes.com/2009/07/08/arts/television/08fight.html
en.wikipedia.org/wiki/Portal:Computer_graphics

Garcia, Fall 2010
 CC BY NC SA

UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (20)

How : Motion Capture

- Actors in MoCap suits
- Motions recorded, put in “motion libraries”
 - E.g., running, throwing, passing, tackling
 - Can be edited/cleaned
 - Motion synthesis also
- Challenges
 - Motion “blending”
 - Non-“sliding” feet
 - UC Berkeley Research!

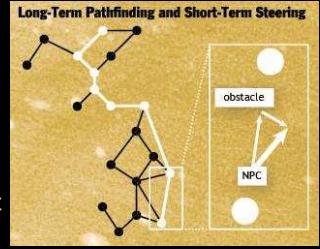



en.wikipedia.org/wiki/Motion_capture
www.phasespace.com

Garcia, Fall 2010
 UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (21)
 CC BY NC SA

How : Artificial Intelligence

- Range of intelligence
 - Low: simple heuristics
 - High: Learns from player
- Dynamic difficulty
 - Must hold interest
 - “Simple to learn, difficult to master is the holy grail of game design.”
 - Cheating AI (e.g., racing)



www.businessweek.com/innovate/content/aug2008/id20080820_123140.htm
en.wikipedia.org/wiki/Dynamic_game_difficulty_balancing
en.wikipedia.org/wiki/Game_artificial_intelligence
queue.acm.org/detail.cfm?id=971593

Garcia, Fall 2010
 CC BY NC SA

UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (22)

Video Games : Good (Serious Games)

- Simulations for training
 - Flight simulations, combat, medical training
- Games w/a Purpose
 - A game to do useful stuff, hard for computers
 - Luis von Ahn ... gwap
 - ESP : Label images fastest
 - Gender Guess
 - Popvideo : label video
 - Matchin : Pick best images

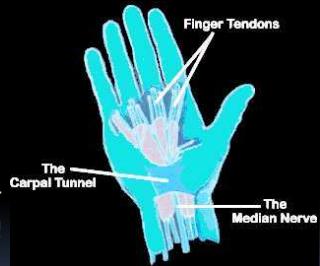



en.wikipedia.org/wiki/Serious_games
en.wikipedia.org/wiki/Game_based_learning
gwap.com

Garcia, Fall 2010
 UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (23)
 CC BY NC SA

Video Games : Bad (RSI, addiction)

- Gamers Thumb
 - Caused with too much use of gamepad
 - I suffered this in 1980s!
 - Solutions?
 - Break timers, rest
- Video game addiction
 - Impulse control disorder
 - Stanford: yes, addictive!
 - “Gamers Wife”
 - Online gamers anon



en.wikipedia.org/wiki/Video_game_addiction
en.wikipedia.org/wiki/Repetitive_strain_injury

Garcia, Fall 2010
 CC BY NC SA

UC Berkeley CS10 “The Beauty and Joy of Computing” : Video Games (24)

Video Games : Ugly (Violence)

- Violent video games
 - Increase aggression, decrease "helping"
 - Others found no link
- High-profile incidents
 - Columbine kids loved the Doom video game
- Ratings help
- Games "folk devil"
 - Billions \$, kids at stake



en.wikipedia.org/wiki/Video_game_controversy
www.apa.org/science/psa/sb-anderson.html

Garcia, Fall 2010
CC BY NC SA

Future of Video Games

- Media producers connecting assets
 - Disney, Lucas big players
- Controllers and sensors expand
- Games on Demand
 - OnLive
- Brain-Computer Interface (BCI)
 - Invasive and Non-



en.wikipedia.org/wiki/Brain-computer_interface

Garcia, Fall 2010
CC BY NC SA

The Fundamentals of Game Design

October 12th, 2010

I got a request via Twitter for this old essay which had fallen off the Internet, so I am posting it here. This was originally written for Metaplace users... there is nothing here new to anyone who has followed the blog for a while, but since it was requested, here it is.

The fundamentals of game design

Starting out creating an interactive experience, of any sort really, can be rather daunting. In this tutorial, we'll run through the basic components of a game, so we can get a handle on what the next steps are when you make the jump from the training tutorials to your own projects.

Often people have trouble when conceptualizing a game. The idea, after all, is often the easy part. It's actually making it, and figuring out where to start, that is the hard part.

A friendly warning, though! Just like writers have different ways of working, and some composers write music in their head and others at an instrument, different game designers are going to have different ways of working. Some work better "in the code" and others like doing everything on paper beforehand. Some think in terms of story and narrative, and others are systems designers first and foremost. So this tutorial may actually run a bit against the grain for you, depending on your natural temperament.

In what follows, I am going to use the language of games, but really, every piece of advice in this article applies equally if you are designing any sort of interactive project whatsoever. So just because I say "game" in what follows doesn't mean this article won't be useful to you when you start making a classroom experience or a chat room or some other application.

The components of a game

The first thing to understand is that games are made out of games. A large game is actually composed of minigames. Even a small game is built out of very very simple small games. The smallest games are ones that are so simple and stupid, you can't lose. You can think of this as "game atoms," if you like.

For example, in the classic puzzle game Tetris, the basic game is beating your high score. To beat that game, you have to master the game of forming lines. There's actually multiple variants there, because you have to learn the games of placing all the different sorts of blocks. And finally, the simplest game is rotating a block, which is just a button and hard to screw up. So games are built out of games. This brings us to key piece of advice #1:

Advice #1: Design one game at a time.

Turtles all the way down

Even if you are making a complex game, built out of many "game atoms," each atom is a game in its own right, and has to feel fun and satisfying. Even the stupid ones with no challenge have to feel good. Imagine how poor a game Tetris would be if the stupidly trivial game of "press a button and watch the block spin" wasn't satisfying.

Many games are ruined at this very fundamental level by poor design. For example, a bad designer might have decided that a random chance of the block not rotating would make sense. After all, we use random chance in

gambling, board games, and roleplaying games, right? But it would make Tetris unplayable.

Game atoms

OK, so you're going to design one of the game atoms. Luckily, every game atom has the same characteristics:

- A player does something.
- The opponent (which might be the computer) calculates a response
- The player gets feedback.
- The player learns from this feedback, and gets to do something again.

You can think of these steps in very abstract terms:

- Input
- Model
- Feedback
- Mastery

Really, that is it. Let's apply it to our Tetris example again. At the trivial "rotate a block" level, we have

- A player hits a button.
- The computer calculates that this means rotate the L counter-clockwise.
- The player is given the feedback of the block in its new position.
- The player figures out "I bet I can do this with other sorts of blocks too. And there's probably a rotate clockwise button somewhere. Rotating is my goal!"

Advice #2: make sure the controls match up well to what the player is attempting to do

At a more advanced level we have

- A player can rotate left, right, drop a block, glance at the next block, etc. Lots of choices.
- The computer is going to take its turn and move the block further down regardless, or spawn a new block of a random shape if there isn't one.
- The block moves down. Maybe it completes a line, maybe it doesn't.
- The player says "aha! Completing lines is my goal, and different shapes help or hinder that!"

Notice that if any of these four steps is poorly chosen, the whole game sucks.

- A player moves the mouse.
- The computer figures this means rotate a block.
- The player is not shown the block, but instead a stock quote.
- The player is baffled and quits.

Advice #3: make sure the player can actually learn from the feedback you give them.

Where does the fun come from?

The fun comes from the mastery process. But what the player is mastering is the model. All games are mathematical models of something. We often speak, for example, of Chess being like war (we actually speak of lots of games as being like war!).

Even games like Tic-Tac-Toe are expressible as math puzzles. Games of resource management over time (like an RTS or Civilization) are exercises in calculus. RPGs where you make choices in character building are actually examples of exploring possibility spaces searching for local maxima... games lie to us all the time about what they are really about.

Advice #4: try to stop thinking about what your game looks like, and think about what it is actually modeling

The underlying math

All this sounds incredibly geeky, but you don't have to be a math geek to enjoy games. The trick is to make the pill go down easy. And the fact is that some math problems and models are more interesting than others. Nobody is that interested in a game that pushes you to solve " $2 \times 5 = ?$ " over and over. It has to be a sort of problem that you can come to again and again, and explore possibilities looking for alternate solutions and paths.

This means that there's a specific and highly varied set of problems that make for good games. In math, a lot of these problems are what is called NP-Hard problems. You don't need to dig into higher mathematics to be a good game designer, though. Instead, you need to ask yourself a basic set of questions:

- Where?
- When?
- How?
- What?
- With?
- For?
- Few?
- Phooey.

This list seems facetious, but it's a shorthand way of asking yourself the following questions about your game atom:

- Do you have to prepare for the challenge?
- ...where prep includes prior moves? ...and you can prep in multiple ways?
- Does the topology of the space matter?
- ...does the topology change?
- Is there a core verb for the challenge?
- ...can it be modified by content?
- Can you use different abilities on it?
- ...will you have to in order to succeed?
- Is there skill to using the ability?
- ...or is this a basic UI action?
- Are there multiple success states?
- ...with no bottomfeeding? ...and a cost to failure?

You have to answer yes to all of these for your game atom to be fun. And yes, we mean every atom in the game has to meet these criteria.

Advice #5: check this list for every action a user can take, every decision they make

How do you get there?

There's really only one way, right now. You prototype and iterate. Don't get hung up on the visuals, except for worrying about whether you are giving enough feedback. The best games can be played using sticks and stones. (You can play most roleplaying games with pretty much any random chance generator, for example).

You can prototype with all sorts of things. I have a "prototype kit" because I often prototype using physical objects before going into the code. It consists mostly of stuff that I can pick up at a craft store:

- Two decks of regular cards.
- One deck of Uno cards.
- One Go board.
- One Checkers board.
- A half dozen six-sided dice.
- One full set of polyhedral dice.
- A large stack of differently colored index cards.
- Twelve pounds of differently colored beads. Go to the pottery aisle at your local craft store — these are the kind that get put in fish tanks and potted plants. It's a bit more than a buck for a pound of one color.
- Wooden pieces, also from the craft store. These are found in the aisle with the clock faces:
 - wood cubes, various sizes
 - colored flat squares, three sizes
 - dowel rods
 - 'pawn' pieces
 - wooden chip (circles)
 - assorted circles, hexagons, stars, etc
 - Blank wooden clock faces that you can draw boards on.
- Wood glue
- Dremel tool
- Square glass chips (also from the craft store, asst colors)

But even that is overkill for most people.

Advice #6: watch others play your game

You'll quickly see where you didn't provide enough feedback, or where they can't figure out the underlying model.

A final thought

Fundamentally, never forget that if you want to design, you have to just go do it. The only way to get better at it is to keep doing it, because gamemaking is in itself a great and varied game to play.

Article copied (and gently modified) from Raph Koster's Website located at <http://www.raphkoster.com/2010/10/12/the-fundamentals-of-game-design/>

The 13 Basic Principles of Gameplay Design

By Matt Allmer

Gameplay design is chaotic and full of frustrations and contradictions. More often than not, the request is to come up with something guaranteed to be successful. This condition steers solutions towards the established – which means solutions that have been done before.

But in the same breath, the product must separate itself from the competition or stand out in some way. This immediately pulls the designer in conflicting directions.

Then, whatever the solution, it must fit within the confines of the project's resources. Not to mention scheduling pressure and strategy changes coming from executive positions.

Hup hup! No time for analyzing the previous paragraph! We've got a title to ship! Never mind your lack of proper tools! Quit your sniveling! Don't you know?

Game design is like sailing a ship while still building the hull! Jump out of a plane while still sewing your parachute and you'll get a good sense of pace in this business. The horse is never put before the cart. We race them side-by-side to see which one wins!

With so much urgency, conflict and uncertainty, there must be an anchor somewhere. Call me boring, but I'm a fan of preparation and established fundamentals. They give me a better understanding of which rules I can break, and which rules I should think twice about.

I took a traditional animation class in college and on the first day, the professor handed out the "12 Principles of Animation", introduced by Frank Thomas & Ollie Johnston. If you're not familiar with these two, they were part of the Nine Old Men: The legendary Disney animation crew responsible for the studio's timeless classics, such as, Snow White, 101 Dalmatians, Bambi, Sleeping Beauty, and others.

At first, these 12 principles were difficult to fully grasp. However, by the end of the semester, I noticed the more principles I applied to my work, the better the animation. Remembering that experience, I think to myself, "By George! Game design should have something similar!"

So, George and I scoured the Internet. Unfortunately, I was disappointed after finding so many disjointed theories, strategies, approaches and creeds. There was a lot of broad subject matter like theories on fun, rewarding players' choices, controlling thought activity, mental multi-tasking... and calls to "simplify" (whatever that means. I'm a designer for crying out loud).

I also found principles so apparent, Captain Obvious would roll his eyes: "know your audience", "don't break the player's trust", "give players choice", "know thyself", "one mechanic in the engine is two in the bush". Alright, the last two were made up, but nothing I found really did it for me.

I was perplexed. None of what I found would help a designer on a day-to-day basis. So George, Captain Obvious and I have decided to throw our proverbial hat into the muddled picture. (And quick! For god's sake, before I collect any more metaphorical personalities!)

The 12 Basic Principles of Animation was my starting point. I took the commonalities and added to them based on what I've identified as the different compartments of gameplay design. You'll notice some are described similarly and some even have the same name, but all apply to gameplay.

The purpose of these principles is to cover all your bases before presenting your designs. You might have a principle fully covered in the beginning, but these principles may spark a thought later when circumstances present a new opportunity. Think of this as a reference sheet. And now, without further ado...

Direction

The first three principles have to do with leading and directing the player's experience. Even though this medium is heavily based on personal, interactive discovery, it is still an artistic medium.

Do not underestimate the importance of artistic direction. Just as a painting leads the eye, a book leads the imagery, a film leads the narrative, so too must a game lead the interactivity.

1. Focal Point

Never allow the player to guess what they should focus on. At the same time, always allow secondary subject matter, but it is the designer's job to clearly provide the primary focus at all times. This applies to both visual and visceral aspects of gameplay.

Level design example * Creating clear, apparent lines of sight.

System design example * Clearly defined plot points and objectives during game progression/user experience.

2. Anticipation

Time is needed to inform the player that something is about to happen. Always factor in Anticipation when designing and implementing events and behaviors.

Level design example

- A train sound effect occurs before player sees train.
- System design example
- An energy charge builds before the lightning attack occurs.

3. Announce Change

Communicate all changes to the player. This short step occurs between Anticipation and the event itself.

The important part to remember is maintaining a hierarchy of notable changes.

A good rule of thumb is degree of rarity. If a change occurs a hundred times in an hour, the announcement may not be required. However, if the change occurs five times throughout the entire game experience, a number of visual cues could be needed.

This principle is so obvious, it can be taken for granted and sometimes overlooked. Be diligent in knowing what changes the player should be aware of at the correct time and on the correct event.

Level design example * "Cast-off" animations trigger for NPCs when the player's character boards the ship.

System design example * An on-screen notification occurs when quest criteria have been completed (i.e. "Slay 10 goblins for Farmer Bob")

Behavior

These next four principles address the very important aspect of behavior. This tackles the player's expectations, both conscious and unconscious. This is where common design theories are addressed such as player choice, reward

and payoff, etc. These principles are also broader, so they can be applied to additional types of design like UI and story...

4. Believable Events and Behavior

Every event or behavior must occur according to the logic and expectations of the player. Every action, reaction, results, emotion and conveyance must satisfy the players' subconscious acceptance test.

Level design example * Place destructible objects near an explosive object. This way, the explosion looks more believable.

System design example * Weaker enemies run away when the advantage shifts in the player's favor.

UI example * HUD elements are affected when player's mech is near death.

Story example * Villagers are more upbeat and react positively after the player has slain the dragon.

5. Overlapping Events and Behavior

Dynamic is lost if only one change occurs at a time. Discover the right amount of events to occur at any given moment of time.

Level design example * Providing the player the ability to build from an appropriate list of structures.

System design example * The linebacker points to direct fellow players, the defensive end shifts over, the quarterback points and calls out football jargon and the crowd cheers louder because it's third down. All this occurs before the snap.

UI example * Points accumulate in the score while each kill is individually tallied on screen.

Story Example * Multiple plot points are at the forefront of the narrative experience. Example: the king is on his deathbed while his war is being waged and he has yet to announce an heir – all while an unknown saboteur orchestrates a military coup.

6. Physics

The player's primary logic operates within the known possibilities of physics. Keep in mind gravity, weight, mass, density, force, buoyancy, elasticity, etc. Use this as the starting point, but do not be limited by it.

Level design example * Ensuring a hole in the floor is the correct size for the correct purpose. Whether it is part of the path of level progression, or simply for visual aesthetics.

System design example * A spark particle effect occurs when the player's vehicle scrapes the side of the concrete wall.

UI example * The GUI's theme references scrapbook elements. In which case, animated transitions, highlights, etc. follow the physical characteristics of paper.

7. Sound

Ask yourself, "What sound does it make when _____ happens?" "Is the sound appropriate?" "Is the sound necessary?" "Does it benefit the experience or hinder it?" If players close their eyes, the sound alone should still achieve the desired affect.

It's debatable whether this principle should be included since Sound Design can be considered separate from Gameplay Design. I've included it because sound is crucial and can easily be neglected. The more it is considered,

the better the experience is for the player.

Level design example * Flies in swamp level make a sound when close to the camera.

System design example * A proximity system where sound effects volume fluctuates depending on distance of game assets.

UI example * Only visually prominent graphics have sound effects attached to them, so as not to muffle the auditory experience.

The next three principles individually touch on other major design components.

Progression

8. Pacing

Keep in mind the desired sense of urgency, the rate in which events occur, the level of concentration required and how often events are being repeated. Spread out the moments of high concentration, mix up the sense of urgency, and change things wherever possible to achieve the proper affect.

Level design example * Create areas for the player to admire the expansive view, versus areas where the player feels claustrophobic.

System design example * Create long, powerful attacks versus short, light attacks.

Environment

9. Spacing

Understand how much space is available both on-screen and in-world, recognize the spatial relationship between elements and take into account the effects of modifying those spaces.

Level design example * Lay out the appropriate amount of space for the appropriate number of enemies to maneuver correctly.

System design example * When an AI character moves through a bottleneck area, walk loops switch to standing idle when the AI character is not moving forward, to show that the character is “waiting” to move through the narrowed space.

Method

10. Linear Design versus Component Breakdown

Linear Design involves solving challenges as they come. All solutions and possibilities hold the same institutional value. Focus can be lost with this method, but it provides creative and spontaneous solutions.

Component Breakdown involves systemic categorization and forming a logical hierarchy of all solutions. This method can restrict innovation but preserves clarity of primary design objectives.

This principle does not mean designers must choose one or the other. There are times during development where one method is more appropriate than the other.

For instance, pre-production provides plenty of time for breaking down a sequence of events. However, when the publisher drops a “must have” change after pre-production, linear design can provide an acceptable solution quickly.

Level design example * Typical blocking of level geometry in an early stage of development, versus adjusting a small area of the same level to implement an idea that wasn't thought of until later.

System design example * Identifying all major systems (combat, AI, input, etc), and progressively filling in various levels of detail versus conceiving the first couple of levels and extracting possible systems based on a linear player experience.

Foundation

The final three principles mark the foundation of gameplay design, which are listed in reverse order of importance. These should be a surprise to no one.

11. Player

How does the player factor into this? How does the player interact with everything that has been designed? More than just device input, address how the player contributes to the experience. If it's a good idea and you're able to convey it correctly but the player is not into it, change it or scrap it!

Level design example * Setting up the player in hopes of making them jump out of their seat.

System design example * Orchestrating progression so that the player feels empowered, determined, anxious, etc.

12. Communication

Is the appropriate team member correctly aware of the objective? Are the appropriate developers clear on the solution? If it's a good idea but you can't communicate it correctly, it might as well be a bad idea because it's very likely to be received as such.

Level design example * Using the elements of the environment so the player is compelled to travel in the correct direction.

System design example * Using visual cues so the player learns when to punch rather than kick, jump rather than strafe, etc.

13. Appeal

When addressing anyone, ask yourself, "Does this draw the audience in?" This applies to (but is not limited to) the player, the spectator, your fellow developers, the publisher, and their marketing team. If it's not a good idea, there's no need to continue until it becomes a good idea or is replaced by something better.

Level design example * Running down the street is not fun, but running down the street while being pursued by government secret agents is.

System design example * Punching can be fun but when the camera shakes on impact, it's even more fun.

Conclusion

So, there you have it. These principles have noticeably improved my designs and forced me to think of components from all angles. I thoroughly believe they will give you an edge on all those impatient carts. So, stick that in your horse and race it!

Article located at http://www.gamasutra.com/view/feature/132341/the_13_basic_principles_of_.php?print=1

This is a test/project grade.

Portfolio Project: Programming Learning Objectives

The Programming Portfolio Task addresses the following CS Principles Learning Objectives:

- 1: The student can use computing tools and techniques to create artifacts.
- 2: The student can collaborate in the creation of computational artifacts.
- 3: The student can analyze computational artifacts.
- 4: The student can use computing tools and techniques for creative expression.
- 5: The student can use programming as a creative tool.
- 8: The student can develop an abstraction.
- 17: The student can develop an algorithm.
- 21: The student can explain how programs implement algorithms.
- 22: The student can use abstraction to manage complexity in programs.
- 23: The student can evaluate a program for correctness.
- 24: The student can develop a correct program.
- 25: The student can collaborate to solve a problem using programming

Task

This portfolio entry includes work on a team (usually 2 members) and as an individual.

Collaborative portion

- Collaborate with a partner to identify an area of focus or subject of your choosing in which you can write and reflect on several different programs that solve problems or do something that is personally relevant to you. You will need to create programs with a purpose.
- The discussion below uses the term area to refer to the subject or focus you have chosen as this first part of this portfolio task.
- The area you choose should be one that allows you and your partner to write programs that go beyond simple explorations.
- Work together to write a program to solve a problem in the area you chose or to illustrate features and characteristics of the area.

Individual portion

Working alone, write another program to solve a different aspect of the problem or to illustrate some other feature or characteristic of the area you chose.

Working alone, produce a single piece of writing that reflects on each of the following:

- the collaborative experience,
- the program that resulted from the collaboration, and
- the program you created individually.

This reflection piece should include an explanation of how you see the programs as part of the area you chose.

Prepare and submit the following

The program you developed collaboratively with your partner within your area. You and your partner will each submit this same program. * Please turn in your brainstorming and storyboards as well.

The program you developed independently within your area. * Turn in your storyboards as well

An individually written description and reflection document as described below in the Requirements section. Each group member must write her/his own reflection.

Presentation

Use PowerPoint or Prezi to create your presentation. Each group member should participate in the presentation

Group Portion

- Discuss your chosen area of focus/subject
- Discuss your chosen target audience
- Discuss the program your group created
- Purpose of the program
- How it might be used by your focus group

Individual Portion

- Discuss how your individual program could be used to solve a different aspect of the problem or to illustrate some other feature or characteristic of the area you chose.
- Each group member should turn in the presentation

Requirements

You must work with a partner on this project. You will collaboratively identify an area and will collaboratively develop a program within that area. You and your partner will work with your instructor to identify a suitable area that lends itself to creating programs that are reasonably complex.

Each program must include each of the following:

- sequencing,
- selection (decision statements - If),
- abstraction (your own procedures), and
- either iteration (looping) or recursion.

You are free to determine the purpose of your programs, but your programs should leverage the richness of the area you have chosen. For example, your programs might solve a problem or create something creative or artistic (though you are not limited to these two examples).

Furthermore, your programs should demonstrate the complexity of the programming environment/language you are working in.

The programs you and your partner produce individually must be different, both in content and purpose, from one another and from the one you write collaboratively. All the programs must be relevant to the area you chose.

Your individually written description and reflection document has a maximum length of 400 words.

Your individually written reflection document should include the following:

- A brief description that identifies the area in which you and your partner chose to work, as well as the problem you attempted to solve or the features and/or characteristics you attempted to illustrate.
- A brief description that describes the additional problem that, working alone, you attempted to solve or the features and/or characteristics you attempted to illustrate.
- A description of why the programs are relevant to the area you chose.

An evaluation of the correctness of both your own individual program and the program you produce with your partner. This evaluation should include

- a description of the algorithms these programs implement,
- an explanation of how these programs function at a detailed level, and
- a justification of why these programs perform correctly.
- A description of the collaboration process you used to create your shared program.
- Describe in what ways this process was or was not appropriate for the creation of your shared program.

Turn in

Every team member should turn in all documents.

- Group Planning - Brainstorming, Storyboards (10 pts)
 - Group Program (25 pts)
 - Individual Planning - Storyboards (10 pts)
 - Individual Program (25 pts)
 - Individual Description/Reflection document (10 pts)
 - Group/Individual Presentation (20 pts)
-

Unit 12: Higher Order Functions

Learning Objectives

Readings/Lectures

- Lecture Slides 12.01: Higher Order Functions ([/bjc-course/curriculum/12-higher-order-functions/readings/01-higher-order-function-slides.pdf](#))

Labs/Exercises

- Lab 12.01: Encryption and Decryption ([/bjc-course/curriculum/12-higher-order-functions/labs/01-higher-order-functions-exercises](#))
-

Higher Order Functions

Computer Science Principles

Handy Blocks For This Section

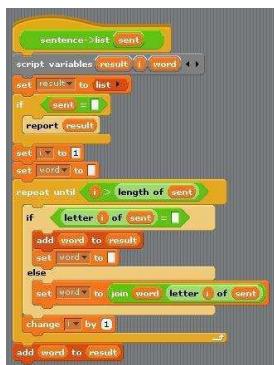
- Sentence->List

- Puts each word of a sentence into a list

- List->Sentence

- Takes the words from a list and builds a sentence

Sentence->List



List->Sentence



Adding Tools

- Need to add more tools so you will import the Tools sprite. How?
- Method 1:
 1. BYOB, go to "File" on top and click on "Import Project".
 2. In the "Import Project" window, locate your BYOB folder and find the "tools" file. Click on the file, then press OK.
 3. After checking that the new blocks are in their palettes, delete the tools sprite in bottom right

Adding Tools

- Method 2:

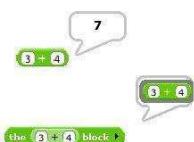
1. Open BYOB
2. Locate your BYOB folder (if you didn't open BYOB from there) and find ToolSprite.ysp.
3. Now simply click and drag ToolSprite.ysp anywhere in the opened BYOB window
4. You will now see all the new blocks that were imported from the tools sprite in each of the palettes, and you can read the description of each of them if you like in the workspace.
 - After you are done, you can delete the sprite that was created.

Higher Order Functions

- Blocks & scripts can be used as **data**.
- How?
 - Put the block or script into the program used as an *action*
 - BYOB will perform that action
 - Need to **encapsulate** that block or script inside another special block called a **wrapper block**.
 - Encapsulate just means to “wrap around or wrap up”

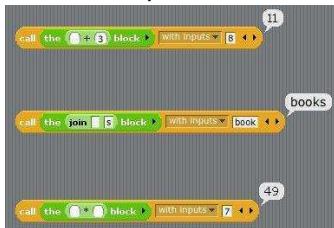
The Block

- **The block** takes a block as input, and reports the block itself.
- Compare these two interactions and you'll see what that means.



The Call Block

- We can take a block with *unfilled* input slots, and **call** it with the necessary input values provided.
- To call a block with one or more input values, click the right-facing arrowhead at the end of the **call** block.
- Here are some examples



Wrapper Blocks

- **BYOB Wrapper Blocks**
 - The block
 - The script
- We will focus on The Block right now.
- Find them in the Operator menu



The Call Block

- So why? We can **call** the function represented by the block — to run, evaluate, or call the block.
- This is where the **call** block comes in.
- The **call** block (Control palette) takes an encapsulated block and evaluates it.
 - If we put the block of $1+1$ into the **call block**, $1+1$ should now be evaluated and return 2.



Higher Order Functions

- **Higher Order Functions (HOFs)** are “higher order” because they are functions that take another function as an input.
- Here's a silly example just to show what we mean.
 - The function **call-with-5** takes a function as input, and reports the value you get by calling that function with the input value 5.
 - You can use any function as the input.



Higher Order Functions

- It turns out that higher order functions are particularly useful along with lists.
- In this lesson we will mainly use three HOFs that take a function and a list as inputs: **Map**, **Keep**, and **Combine**.
- They are found in the "Variables" palette, with the other list blocks.
- Even though we emphasize them, keep in mind that they aren't the only higher order functions; in fact, you can write your own.

Higher Order Functions

- All of the three main HOFs take a function and a list as inputs, and the function will somehow be applied to the list items.
- The result varies depending on the HOF.
- In BYOB, HOFs are all reporters



Higher Order Functions

- If the reported value is a list, it's a newly constructed list; these blocks do **not** modify their input lists.
- This style of programming, in which existing values are not modified, is called **functional programming**

THE MAP BLOCK

The Map Block

- You decide that you want take the first letters of a couple words using the "letter _ of __" block in a list.
- To start with a simple case, suppose you have a list of two words.
- You could apply the "first" block to each of them and make a list out of the results



The Map Block

- Similarly, here's the version for three words in a list

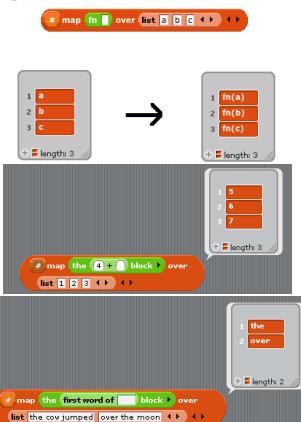


- This approach, though, would get very redundant if you had a list of five words - you'd have to make blocks specifically for the case of five words. You don't want to keep making blocks for a specific case.
- Of course, you can use recursion to solve this problem. But you can also use the Map block.

The Map Block

- **Map** takes a reporter block and a list as inputs.
 - It computes a function of each item of the list and report a list of resulting values.
 - The first figure is an abstract representation of how map would apply to an arbitrary function **fn**; the others are examples of how to use **map**.
 - You'll notice that at the left end of the map block is an orange circle containing a number sign.
 - Ignore that for now; it's a rarely-used feature that we'll get back to later.

The Map Block



Transforming the Beatles

- Make a "transform-beatles" block that takes a reporter as an argument, applies it to each of the Beatles (John, Paul, George, Ringo), and returns the result in a sentence.



Transforming the Beatles

- Use the "list->sentence" block, and make sure you use a *list* in the script that defines this block.



- Note that transform-beatles does *not* take a list or a sentence as an input.

- We can use this to "amazify" the Beatles!

Transforming the Beatles

- Let's create the Amazify procedure first.



Transforming the Beatles

- Now, let's drop the Amazify procedure into "the Block" so we can call it when we need it.



Transforming the Beatles

- Create a reporter block called transform-beatles with an argument called block.
 - Why call it block? Because our Amazify block is going to be the argument.



Transforming the Beatles

- Let's drop the map (block) over block into a list to sentence block so that we get the output as a sentence.



- All that is left is dropping our block into a report block and adding it to the command.

Transforming the Beatles

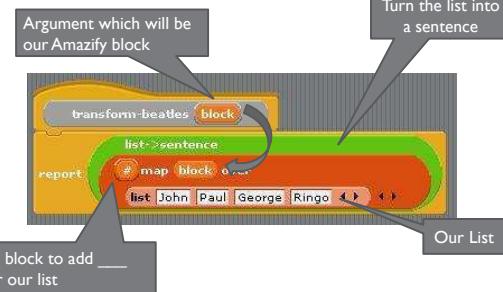
- Now let's create the transform-beatles script.
 - Let's think about what we want to happen... We want to add or map "the amazing" to each name in our list. So let's grab the map block.



- Add block (our argument) and a list with all of the names.



Transforming the Beatles



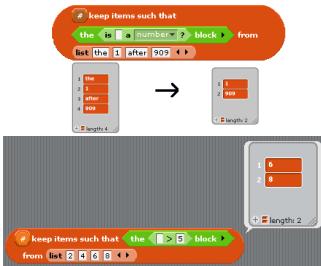
Map Exercises

- Make the "first-letters" block that was introduced using map.
- Given any list of numbers as an input, use map to add each of those numbers in the list by ten.
- Given any list of words as an input, use map to add a letter "s" to the end of those words in the list.

THE KEEP BLOCK

The Keep Block

- **Keep** takes a predicate block and a list as inputs, and reports a new list containing the subset of that list for which the predicate reports TRUE.



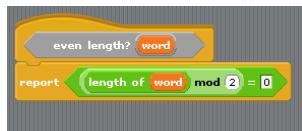
Choose Beatles

- Make a "choose-beatles" block that takes a predicate block and returns a sentence of just those Beatles (John, Paul, George, and Ringo) that satisfy the predicate.
 - Note 1: Your task is to write CHOOSE-BEATLES, not EVEN LENGTH?.
 - Note 2: CHOOSE-BEATLES doesn't take a list of Beatles as input; its only input is the predicate block that specifies which of them to keep in the result.



Choose Beatles

- Let's first create the Even Length? Procedure
 - It should report true or false (predicate)
 - Remember mod is remainder division; so if the length of the work mod 2 = 0 then it has an even number of letters.



Choose Beatles

- Now let's create the Choose-Beatles reporter.
 - Add an argument called predicate. This is where we will drop the Even Length? Block.



Choose Beatles

- Grab the `keep` block and drag the argument "predicate" into the empty predicate spot.
- Drag a `list` block into the empty list spot and add the names.



Choose Beatles

- Since we want our answer as a sentence, grab a `list->sentence` block from the operators area.
- Drag the `keep` block into empty list spot



Choose Beatles

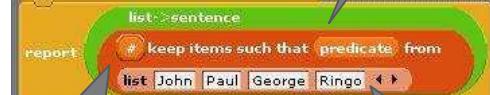
- Grab a Report block (Command) and drag the list->sentence into the empty report space then click it into the header block.



Choose Beatles

Argument

Turns the list into a sentence



Combine

° COMBINE

- **Combine** also takes a two-input reporter block and a list as inputs.
- Combine uses the two-input reporter block to combine items from the list one by one into a single value.
 - For example, the block shown below starts with 4, then computes $4+5=9$, then computes $9+6=15$.

Combine

- Unlike MAP and KEEP, COMBINE really makes sense with a small number of reporter blocks as input, because the operation has to be associative (giving the same answer no matter how the operands are grouped).
- The blocks you'll use often with COMBINE are +, * (but not — or /), MAX, MIN, JOIN, and JOIN WORDS (which is in the tools package).

Letter Count

- Make a reporter block "letter-count" that takes a sentence and reports the total number of letters in the words of the sentence (not counting spaces between words).



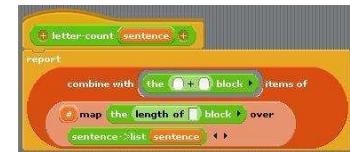
Letter Count

- You will need to add the length of each word over our string.
- First grab a map () over () block.
- You will need to get the length of [] block (to get the length of the word) then drop it into a “the () block” block.
- Get a sentence->list block with the sentence argument to drop into the empty list spot.



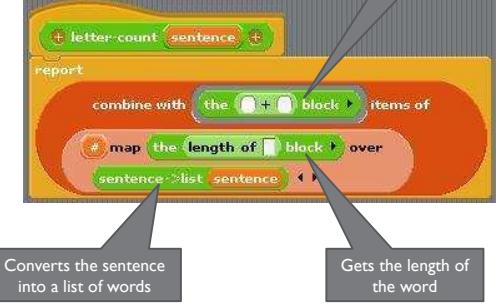
Letter Count

- Grab a combine with block
 - Grab a () + () block and drop it into “the block” block
 - Drop your map block into the empty “items of ()” space.
- Grab a reporter and drop in your block.



Letter Count

Calculates the running total

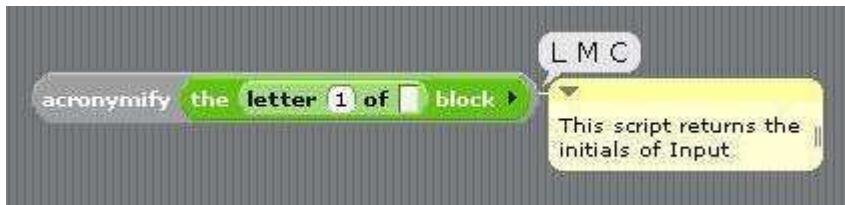


Curriculum (/bjc-course/curriculum) / Unit 12 (/bjc-course/curriculum/12-higher-order-functions) / Lab 1 (/bjc-course/curriculum/12-higher-order-functions/labs/01-higher-order-functions-exercises) /

Higher Order Functions Exercises

Map Exercise: Acronymify

Using the transform-beatles block as an example, create a block that will report the first letter of each word to create an acronym or initials.



Keep Exercise: Starts With A

Using the Count "ums" block as an example, create a block that will report a sentence of just the list input that starts with "A".

Hint: You will first need to create a predicate block that determines if the word starts with the letter "A"

Hint: Starts-With-A will take the predicate as its input



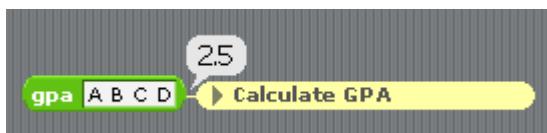
Combine Exercise: Calculate GPA

Using the Letter-Count block as an example, create a block that will calculate your GPA given a sentence of letter grades.

Hint: A = 4.0, B = 3.0, C = 2.0, D = 1.0

Hint: First you will want to turn the sentence of letter grades into a list.

Hint: You will need to create a reporter block that will report the numeric equivalent of the letter grade



Object Oriented Programming with Game Maker

Learning Objectives

Download

- Game Maker Lite for Windows (<http://www.yoyogames.com/gamemaker/windows>)
- Game Maker Lite for Mac (<http://www.yoyogames.com/gamemaker/mac>)

Readings/Lectures

- Official Game Maker Tutorial (<http://sandbox.yoyogames.com/make/tutorials>)
- Resources for Teachers (http://wiki.yoyogames.com/index.php/Information_For_Teachers)
- More Game Maker Resources (<http://gamedev.edublogs.org/2012/12/07/game-maker-resources/>)
- Game Maker Tutorials (<http://www.screencast-o-matic.com/channels/c661FeVje>)

Labs/Exercises

- Tutorial 13.01: Designing Games (/bjc-course/curriculum/13-object-oriented/labs/01-designing-games.pdf)
- Tutorial 13.02: Your first game (/bjc-course/curriculum/13-object-oriented/labs/02-your-first-game.pdf)
- Tutorial 13.03: Maze game (/bjc-course/curriculum/13-object-oriented/labs/03-maze-game.pdf)

Sprites and Resources

- Resources for 13.02 (/bjc-course/curriculum/13-object-oriented/labs/first-game.zip)
- Resources for 13.03 (/bjc-course/curriculum/13-object-oriented/labs/maze-game.zip)

Designing Good Games

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: no specific version of Game Maker

Level: Beginner

When Atari produced its first game console in the seventies it was not very popular. This changed drastically when the game *Space Invader* was created and bundled with the console. Within a short period of time Atari sold a huge number of consoles. The same thing happened when *Pacman* was produced. And for the Nintendo Game Boy *Tetris* was the absolute winner. Why are these games so special that they mean the difference between success and failure of the devices they were created for?



Figure 1. PacMan and all of its clones are still very popular games.

The same applies in PC games. Some games become extremely popular making their creators instant millionaires, while other games, that look almost the same, become miserable failures. And then there is also a large collection of games that you never see because they were cancelled halfway the production and their creators went bankrupt. What makes a game a winner and what leads to failure? This is a very difficult question to answer. It involves many different aspects. In this tutorial we will delve into some of these aspects in the hope it will help

you to create better games. Many elements of this tutorial were based on a paper by Creg Costikyan¹.

What is a Game?

Before talking about good games we should decide what a game is in the first place. There is a surprising amount of discussion about this issue and there are many different definitions. It is easier to say what is not a game. This

A movie is not a game

This is rather obvious, but why? What elements of games are missing in movies? The main difference is that there is no active participation of the viewer in a movie. The viewer does not control the movie and cannot make decisions that influence the outcome of the movie. The same is true for stories and plays in a theater. Also the final outcome of the movie is fixed (even though the viewer might not know it). This is a crucial aspect of movies and plays. People in general don't like plays in which the outcome is not predetermined. In games the opposite is true. People do not like it when the outcome of a game is fixed. They want influence on that outcome. They want to be in control.

A toy is not a game

You play *with* a toy but you do not play *with* a game. You play the game. With a toy there are no predefined goals although during play you tend to set such goals yourself. A number of computer games actually are close to being toys. For example, in *SimCity* or *The Sims* there are no clearly defined goals. You can build your own city or family and most likely set your own goals (like creating the biggest city) but there is not really a notion of winning the game. One could add this (e.g. you could add that the game is won when your city has reached a particular population) but this can be frustrating because it is not a natural ending. This being said, there is nothing wrong with creating a nice interactive computer toy.



¹ Creg Costikyan, I have no words & I must design, Interactive Fantasy #2, 1994. See also <http://www.costik.com/nowords.html>.

Figure 2. Is SimCity a game?

A drawing program is not a game

A drawing program is fun to play with and encourages creativity, but again it has no clear set goals. The user defines the goals and it is the user who decides whether the goals are reached.

A puzzle is not a game

This is a more difficult one. Clearly many games contain puzzle elements. But a puzzle is static, while a game is dynamic and changes in the course of playing it. A satisfying game can be played over and over again and there are different strategies that lead to success.

So what is a (computer) game then? Here is a possible definition:

A computer game is a software program in which one or more players make decisions through the control of game objects and resources, in pursuit of a goal.

Note that the definition does not talk about graphics, or sound effect, or in-game movies. Such aspects obviously do play a role in making nice, appealing games, but they are not the essential aspects of games. Let us look at the different ingredients of the definition in some more detail.

A computer game is a software program

This makes it rather different from for example board games or sport games. It takes away some of the fun of games. There are no pieces to move around and there is no physical satisfaction (although some recent games, like *Dance Dance Revolution* or the games for the new Nintendo *Wii* console involve physical exercise). Also the social aspects are less prominent, although online multiplayer games add a new form of social interaction. But we get quite a bit in return. A software program can much better react to and adapt to the players. Most computer games have a real-time element that is not present in board games. The game continues even when the players do nothing. This can lead to enhanced excitement and a better feeling of presence in the game world. Also computer games can adapt to the players making it satisfying for largely different players, both beginners and advanced. The possibility of having computer-controlled opponents adds many new challenges. Computer games can also be more complex because the game itself can help the players understand the different aspects and teach the player how to play. Finally, computer games can create a more immersive environment by adding wonderful graphics, music and cut-scenes.

A computer game involves players

This is rather obvious. A game is not something to watch. You should be involved in a game. Don't underestimate the importance of the player. Beginning game designers often forget that you make the game not for yourself but for the people that are going to play it. So you always have to think about who they are. A game for children should be rather different than a game for adults. And a game for hard-core gamers should be rather different from a game for less experienced players. You need to pick the correct audience. Bad games are often written for the wrong audience. For example, a very experience flight simulator freak wants to be able to

control every aspect of the plane and wants things to be as realistic as possible. For a player that just wants a bit of quick flying fun this is frustrating and boring and such a player will most likely never get the plane to take off, let alone to land.

Playing a game is about making decisions

The player makes decisions that influence the rest of the game. In fast paced action games such decision typically involve in which direction to move and which weapon to choose for shooting. In complicated strategy games the decisions involve where to build your settlements, which units to train, when and where to attack, etcetera. Of course decisions should have an effect. Surprisingly, in many games the effect of decisions is only marginal. For example, often it does not really matter which weapon to use. This often leads to frustration. Carefully balancing decisions and their effects is crucial for satisfying game play.

Playing a game is about control

The player should feel in control of the game. Not the other way round. Uninterruptible sequences in which the control is taken out of the hands of the player still occur in many games and often lead to frustration. The more freedom there is for the player, the better. There is though a catch here. A game is also about surprises and dramatic effects. Such effects can be created much better if the player is not in control. For example, in a movie, when the main character approaches a door you can let the music rise. The viewer knows that something is going to happen. Together with zooming in on the door, this can create a great dramatic effect. But if the same happen in a game and at the last instance the player decides not to open the door, most of the effect is gone and even becomes absurd. Careful balance of freedom of control and dramatic effect is difficult. (There is another less valid reason for not allowing too much control. More freedom and control for the player makes it more work to create the game.) Whenever you need to constrain the user, try to do this in a natural way. For example, in *Riven* the player moves between different parts of the game world. By letting the user use some kind of train system it is natural that this motion goes automatic and cannot be controlled by the player.

Game objects and resources

In a game you normally control certain game objects, like the main character, units, a car, etc. In some games you can control just one object while in other games, for example strategy games, you can control many different objects. Besides the game objects that the player controls, there are normally many other objects that are controlled by the computer. The game objects the player controls play a certain role in the game. This is an important property. In other programs you also control certain objects, like buttons, but these do not play a role in the program. They are only meant to give certain commands to the program. Besides controlling game objects you must often also manage certain resources. This is most evident in strategy games and simulation games in which you must manage the amount of food, wood, stone, gold, etc. But also in many other games there are resources to manage, like ammunition for your weapons, a shield that can be used a limited amount of time, etc. Careful planning of resources and their use can add

many nice aspects to the game play. The game designer must balance the availability of resource with their need, to achieve interesting game play.

A game needs a goal

This is a crucial ingredient in a game. People want to win a game and, hence, there must be a goal to reach. For long games there should also be sub-goals, like finishing a particular level, defeating a certain monster, or acquiring a new spell. Reaching a goal or sub-goal should result in a reward. Such a reward can consist of a score or some nice movie, but it is better if the reward is actually part of the game play itself, for example a new weapon, some additional useful information, etc. We will talk more about goals and rewards in a moment.

What is a Good Game?

So now we know what a computer game is. But it does not say much about when a game is good. Think about the following computer game:

You have to rescue the princess who is held in a fortress. On the screen you are shown two roads, one leading to a fortress and the other leading to a cave. You have to decide which road to take. You choose the road to the fortress? Congratulations. You rescued the princess and won the game. You choose the other road? Bad luck. You are eaten by the cave monster and die.

If you verify it, this game has all the ingredients described above. There is a player, there is a decision to make, the player controls what is happening, there are game objects (the prince, the cave monster, etc.) and there is a clear goal. But it is obviously a rather boring game. There is no challenge. The game is too easy. So clearly we have to do a better job to make an interesting game.

Reaching Goals

An important part of a game is that there is a goal and the game challenges the player to try and achieve this goal. Actually, there are often many different sub-goals. Goals come in all sorts and shapes. A goal can be to try and shoot an enemy plane, or to finish a level by collecting all diamonds, or to reach the highest score or to finish the game. Clearly some of these goals are short-term goals while others are long-term goals that can only be reached by playing the game for weeks. A good game is filled with these goals and the player should be rewarded when he reaches one of the goals. Rewards give an important additional motivation to try and reach the goals.

Goals should not be too easy to achieve. There must be a challenge. And when the game progresses the goals should become harder to reach and the player has to become better at the game to achieve them. This learning curve is very important. In the beginning the player needs to understand the controls and the mechanisms in the game. This is best done by letting him achieve some simple goals. Later on, the player understands the game better and will be ready for bigger challenges.

Obviously, when goals are hard to achieve, there is a big chance of failure. You have to be careful with failure though. It can easily put the player off, making him stop playing. And that is definitely not what you want to happen. To avoid this it is crucial that, in the case of failure, the player always has the feeling he made a mistake that he could have avoided. It should not be the game's fault that the player lost, but his own. It is one of the aspects that distinguish games like *PacMan* and *Tetris* from other games. You always have the feeling you did something stupid. You can be pretty angry with yourself when it goes wrong and you are determined to avoid this mistake the next time. This feeling keeps you playing the game. On the other hand, consider a maze game in which from time to time at a random spot a flash of lighting occurs, killing you if you happen to be in the neighborhood. In such a game you, as a player, did nothing wrong. You just had bad luck to be at the wrong spot. This is very frustrating. You are not angry with yourself but with the game. And you probably soon stop playing it. Don't think that commercial games are perfect in this matter. Quite some games for example produce enemies at random locations and random moments in time. If you have bad luck they appear at the wrong moment right next to you and kill you.

You should learn from this that you have to be careful with "luck" in your games. Whether the player can achieve a goal should not depend on good or bad luck. Bad luck is of course very frustrating for a player but also good luck does not give the player satisfaction. Imagine that you can be lucky and find a super bomb just before facing the main enemy. Having the super bomb make the fight very simple while not having it makes it a major challenge. With the super bomb the player will not have the feeling he conquered the enemy himself. It would have been much better if the super bomb was always there but the player had to make a difficult move to get it, for example, jumping over a dangerous pit. Now the player has an interesting decision: performing the dangerous jump to make the fight easy, or not risking the fall and fighting the enemy with lesser weapons.

Decisions

As we saw in the last example, creating an interesting decision enhances the game play considerably. In general, decisions are a crucial ingredient of games. The more interesting the decisions, the more interesting the game is. There can be very simple low-level decisions or very high-level strategic decisions.

Let us look at the well-known *PacMan* game. It is packed with decisions. The most important decision that you constantly have to take is which direction to move in. Are you trying to stay as far as possible away from the monsters or are you going after the dots, even if the monsters stay close-by? And will you go to a corner, where you might be caught or will you stay in the center where you can move in more directions but can also be attacked from multiple sides? A second type of decisions lies with the pills you can eat to chase the monsters. When are you going to use them? Do you leave them to the end and only use them to get to the final dots or do you use them early on to clear most of the maze? And if you eat them, are you going to hunt

for the monsters to get extra points or are you going to use the safe time to eat more dots and try to finish the level? And finally there is the bonus item that appears from time to time. You can try to get it for extra points, but you will run the risk of being eaten by a monster.

When there are many decisions to make, like in *PacMan*, the player will make mistakes. In *PacMan* these mistakes are not immediately fatal, but it will require you to work harder to finish the level or to get the highest score. This is important because everybody makes mistakes and you should not be punished too much for such mistakes. In the same way as a reward should be related to the achievement you made, a punishment should be related to the seriousness of your mistake. If the player loses the game, this should be the result of a grave mistake or a series of smaller ones. In such a case the player will definitely feel that he himself is to blame for the loss, and will continue playing to try to do better.

Balance

In a good game different game aspects are balanced. For example, the player should have the weapons with which he can fight the enemies. The weapons should not be too strong. That would make the game too easy. And they should not be too weak because then the player can only survive if he has a lot of luck, and remember what we said about luck before. Balance is difficult to achieve. And players are very clever in finding out where the game is unbalanced and exploit this unbalance, thereby often ruining the fun of the game.

There are three different aspects of balance: balance between players, balance between the player and the game play, and balance between different features in the game. We will discuss each of these below.

Balance between players

If you create a two-player game, you better make sure that the best player normally wins, and not the most lucky one. Imagine a strategy game in which two players compete with each other. As in most strategy games they have to build up a city and for this they need wood. Now imagine there is just one forest in the world and one player starts very close to this forest and the other is far away from it. This gives the first player an advantage that will most likely win him the game. So the game is highly unbalanced.

A game of chess on the other hand is highly balanced. Each player has the same pieces and can make the same move. The only problem is that one player can start and this is actually an advantage in chess. But this is balanced out because in a match each player can start the same number of times.

Chess is a symmetric game. Symmetric games are well balanced. But symmetry is also a bit boring. Imagine that in the strategy game I mentioned the world looks completely symmetrical and each player plays the same race with the same units. That would make the game less appealing. Still it is used rather often. For example, the multiplayer maps in *Red Alert II* are very symmetrical. The real game design challenge is to make a non-symmetrical game that is still rather balanced.

One way of achieving this is to use fake asymmetry. Let me demonstrate this with an example. In our strategy game we let the first player start behind a mountain range while the second player has his city behind a river. The first player we give the ability to create boats while the second player has equipment to drill tunnels. This looks very asymmetric but the tunnels can be used to pass the mountain range and in a similar way the boats can pass the river. So balance is restored again. Many strategy games use some type of fake asymmetry. Races might look rather different but in the end the possibilities are very similar.

Balance between the player and the game play

The game play is there to help the player, not to fight the player. As was said before, the player should lose because he made a mistake, not because he for example forgot the key combination to fire the canon. Careful design of the interaction (the use of the keyboard, mouse, joystick, etc.) is important to avoid this type of problems.

Also you need to strike a good balance between what the player must do and what the game does for him. For example, in most games the player does not need to continuously push buttons to make a game character walk. The game does this automatically for him. But the player must press a button to make the character shoot. In many strategy games, soldiers automatically start attacking enemies that come in close range rather than requesting the player to constantly check on all the units. But the player must decide when to start an invasion into foreign territory. But also well-known games make the wrong decisions here. For example, they force the player to constantly bring food to the troops or they force you to manually withdraw wounded soldiers from the battle. For example, one of the things many people complained about in *Black and White* was that when your people were praying you had to bring them food all the time.

Let us consider another example. In the early adventure games one of the major challenges was to find out where you should click on the picture to get certain things done. For example, to open a door you had to find the secret button to press. Only after pressing on all the 100 stones in the wall you found the one that opened the door. This adds no fun to the game. In modern adventure games the mouse cursor changes whenever you move it over a place where you can click and often a message appears indicating what there is to click on. Good visual cues are also given, for example by giving one of the stone a slightly different color. This will improve the game play a lot. The player still has to come up with the idea that there might be a secret button but once he has that idea it is easy to find the place.

The bottom line is that the player should spend his time and energy on the important aspects, and the game program should do the rest. The game should try to understand what the player wants and take action accordingly, rather than the other way round.

The balance between game features

A game contains many different features: different weapons, different enemies, different units, different roads, all sorts of resources that can be used, and so on. These features result in

decisions for the player: which weapon to use for what enemy, which road to take, how to use the resources, and so on. This makes the game interesting. But you better make sure there are some real decisions here. For example, when your game features four types of weapons, but one is superior to the others, the player will never use the other three weapons once he finds the best one. So there is no decision left anymore. To keep the decisions interesting you should balance the good aspects of the features with the bad ones. For example, the powerful weapon can fire only one shot per second, or the ammunition is more expensive, or it cannot be used in a cave, or one opponent is more sensitive to a particular weapon than another. Use your creativity.

You have to balance the powers of the player with the power of the computer-controlled opponents. When new opponents appear during the game, you should give the player new powers to fight them. But be careful that you don't fall in a well-known trap in which you simply increase the firepower of the player while the opponents get equally stronger. This does not lead to more interesting game play. There is not much difference in driving with a slow car against slow opponents or with a fast car against fast opponents (unless, of course, steering the fast car is more difficult). A key issue here is that the player should improve during the game, not the character he plays (or the car he drives). This is not to say that the character of car should not improve. But the improved character should reflect the improvements in the player.

Don't forget that a player must learn to play the game. That is, the game should start easy with easy decisions for the player to make. When the game progresses and the player becomes better at it, he should get more and more complicated decisions to make. This can be achieved by introducing new features gradually during the game. The features should match the players' abilities. Make sure that there are still new features appearing far into the game. Too many games show all the features in the first few levels after which the game becomes just more of the same. Good games come up with surprises, all the way till the end.

Rewards

You need to reward a player when he achieves a goal. A reward can take the form of a particular score, some nice graphical or musical feature, or items that can be used in the game, like better weapons, power-ups, spells, or knowledge about the game world. The last type of reward is definitely the most rewarding to the player and whenever possible you should try to create this type of rewards. The effect can be permanent or temporary. Temporary rewards are typically given when a player achieves minor goals. It makes the playing easier for a while. Examples of this type of reward are some extra ammunition, or temporary invisibility to opponents. Permanent rewards are given when bigger goals are achieved. For example, you get a new weapon or spell or car. This will change the game play from that moment on, hopefully extending the range of decisions the player can make.

Giving the player the right type of rewards is actually an issue that is harder than you might think. People are picky about their rewards. If the rewards are too small they will not work hard to achieve them. If they are too large they get greedy and want even bigger rewards. It is a well-

known psychological phenomenon that players start expecting rewards and if you somewhere during the game you decide that a particular reward is no longer available they get angry. Let us consider an example of this. If in the first level of the game you give the player a bit of extra health for each opponent he kills, the player starts expecting this. If you decide in the second level that the player should now be more experienced and you stop giving this reward the player tends to be upset and might stop playing the game. It would be better to gradually increase the maximal player health and the damage opponents do, such that the increase in health is not significant anymore. The player still gets his reward but it has less influence on the game play.

You also need to decide whether rewards are predictable or more random. For example, in your game you might give a bonus item for each 50 collected coins. Alternatively, with every coin you collect you have a 1/50th chance of getting the bonus. Even though mathematically equal, the effect of these two choices on the player is completely different. In the first situation, in the beginning the player is not very interested in collecting coins. It will take way too long before it will result in a bonus. This will make the game play less intense so there should be other aspects that keep the player interested, like exploring the environment. But when the number of collected coins approaches the 50 the game plays starts becoming very intense and the player will work very hard in collecting opponents, even those at difficult spots. So there is a high variation in intensity, which is appealing to certain types of players. When the award is randomly there is always an interest in trying to collect coins because it might lead to a reward. So the average intensity of the game will be higher. But there will be no peaks in intensity, which can lead to a more dull game.

Make sure the player notices the rewards he gets and starts understanding why he gets them. If the player does not know the relation between his actions and the rewards he gets this will be frustrating and will lead to less focused game play. So clearly indicate when points are scored or power-ups are obtained. For example, use some sound effect or some graphical effect.

Flow

A game gives challenges to the player and the player develops abilities to conquer these challenges. Challenges can take the form of monsters to beat, obstacles to avoid, puzzles to solve, bases to attack, and systems to master, for example a plane. The abilities a player must develop depend on the game and can for example be reaction speed, strategic thinking, or knowledge. A game is only fun to play when the challenges are in balance with the abilities of the player. While the game progresses the abilities of the player improve and, hence, the challenges should become more difficult. It is the task of the game designer to keep challenges and abilities in balance. This situation is called the *Flow*. When challenges are too hard the player gets frustrated, when they are too easy the player gets bored. There actually is a band in which the game is still fun to play. If you get to the top of the flow you reach a state that is sometimes referred to as *pleasurable frustration*. It is good to let your game from time to time get to this top and then give some easier challenges again. This helps the player to improve his abilities. So the difficulty should zigzag through the flow.

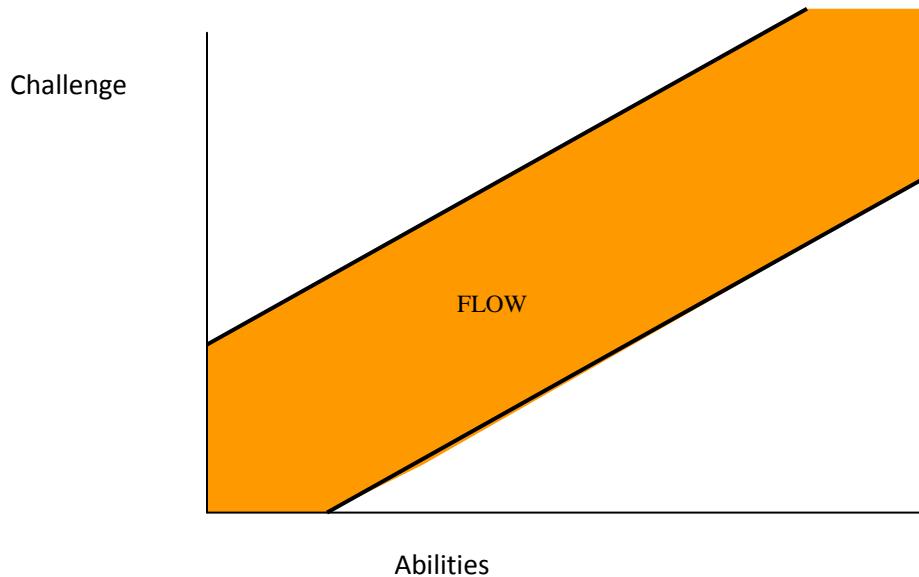


Figure 3. Keeping the Flow.

Keeping a game in the flow is difficult because it depends on the player. The easiest way is to give the player the opportunity to choose a level but this is not very effective, unless there is a big reward in playing on a more difficult level and it is easy to change level during the game. A second option is to let the player skip certain challenges and do alternative ones, better suited to his abilities. But most players tend to take the easier route, even if it leads to boredom. So the best way is to adapt the challenges to the player. Monitor the players behavior (for example how much damage he takes) and adapt the number or (better) quality of the opponents to this. Make sure that the player always progresses but let the reward depend on his qualities.

Presence and Immersion

You might have wondered why we did not talk about graphics yet, or about sound and music. Many people consider them crucial ingredients of a game. New commercial games try to achieve great new graphical effects and hire famous musicians to create the music. So isn't this important? Well, yes and no. If you look at the games available on devices like the *Nintendo Game Boy Advanced* or mobile phones, they have rather poor graphics and the sound is also limited. Still they are great fun to play and many people are addicted to them. On the other hand, some of the best three-dimensional games create a special atmosphere using the right type of music and stunning graphics effects like dripping water, smoke, and flickering torch lights.

The key issue here is *immersion*. Game play is largely enhanced if the player feels immersed in the game; if he feels that he is present in the game world and that his decisions and actions really matter; and if he becomes emotionally attached to the main characters in the game and really wants to help them. Important ingredients to achieve this immersion are the story in and behind the game, the surroundings in which the game takes place, the way the main characters in the game look and behave, the music, and the special effects.

The story

There is a lot of discussion about whether a game needs a story. Popular games, like *PacMan* or *Tetris* do not have a real story (although the designers still gave them some sort of story). And in many first person shooting games, the story is almost always the same: rescue the world from some kind of evil. Most people never read the story and it seems not to influence the way they experience the game. (You are not trying to save the world; you are simply killing the monsters that attack you.) On the other hand, for adventure games the story is crucial. It forms the basis for the puzzles you need to solve, and the story actually helps you solve the puzzles; they often only make sense when being part of the story. Also other games can benefit from a good story; again because they give a meaning to the actions you are performing and deepen the satisfaction when reaching your goals. It leads to *Meaningful Play*. This can be achieved by making sure that the different tasks or levels in the game form a logical sequence and by putting cut-scenes or movies in between them to enhance this storyline. Designing a good storyline with movies, etc. is probably beyond the skills of most beginning game designers, but it is good practice to at least put some logic in the game you are creating and such logic normally comes from a story.

The game world

A game takes place in some world. This world can be presented in exact three-dimensional realistic detail but also in a more abstract or cartoon-like two-dimensional way. Some games just use text and some static images to represent their game world. Designing an interesting game world is an important part of game design. And picking the right type of representation is important too. For a first-person shooter a well-detailed three-dimensional game world with lights, shadows, and special features like mist and water is crucial to give the player the feeling of presence. He has to see what a real fighter would see, otherwise the game becomes artificial. For a flight simulator the world should also look as realistic as possible. For an adventure game a realistic three-dimensional world is not so important. Here it is the story that creates the feeling of presence and this can also be accompanied by simple two-dimensional images. In puzzle games and many arcade games the game world is rather abstract and often two-dimensional. For example, in a scrolling shooter planes don't fly in natural ways nor do the bullets behave natural. And power-ups might float in the air. This is all perfectly acceptable for the player when the game world is rather abstract but would be out of place when the game world would look realistic. So it is really important to adapt the game world to the type of game you are creating.



Figure 4. A flight simulator should be realistic, while a scrolling shooter can be more abstract.

A realistic three-dimensional world can also hamper game play. For example, many strategy games use a form of overhead view of the game world in which you view it under a 45 degree angle (a *isometric* view). This makes it easy to track your units and to quickly see what is happening. You can easily scroll over the world to steer your units in doing the right things. Trying to do the same in a full three-dimensional world is a lot harder. You quickly lose your orientation, and have difficulty in keeping track of what is happening in the world. Moving around is more difficult. Again you must adapt the representation of the game world to the game play that is required.

The main characters

Many games have one or more main characters that the player controls or meets. Like in a movie it is important that the player becomes emotionally attached to these characters. He can hate them and try to kill them or like them and try to help them. So characters and their behavior need to be designed carefully. How again depends on the type of game. For example, in a first-person shooter the player himself is the character. He should fully identify himself with the character. In such a case it is advisable not to give the character a strong personality. This makes it more difficult to identify yourself with him. Or at least give the player the possibility to choose between different characters to pick one that suits him. For third-person games and adventures a strong personality is often important. If done right, the character can get some kind of hero status, like Lara Croft from *Tomb Raider*.

Music

Music and background sounds can play a very important role in immersing the player in the game. Even very soft background sounds can have a dramatic effect in games. For example, dripping water in a cave gives a creepy sound. Rolling thunder can raise the players fear, etc. Background sounds can also provide clues to the player about what is going on. For example you can hear footsteps in the distance or a door that is slammed shut. Modern games use positional sound such that the player also knows where things are happening. Picking the right kind of music for your games is as important as picking the right kind of graphics. A cartoon style game should have cartoon style music. Creepy games should have creepy music, and funny games should have funny music. Better have no music than the wrong kind of music. Modern games

nowadays use adaptive music that changes with the action that is happening. This can further increase the dramatic effect but is definitely beyond the possibilities for beginning game designers.

Special effects

Like in movies, special effects can have an important effect on the player. Some great explosions or sound effects can temporarily highly enhance the game experience. But be careful. The effect soon wears off. After 10 of such explosions you won't even notice them anymore. And they might even become annoying if they hamper the game play, e.g. by slowing down the refresh rate, or distracting the player. For example, some puzzle games have beautiful color changing or animated background. Soon these become very annoying and you really want to switch them off. So don't spend too much time and effort on special effects. Better concentrate on good game play.

Game Genres

Games come in many different types. Over the years a number of different genres have been created. If you are very creative you can try to make a game that is completely new, but if you want to be on the safe side you better pick a particular genre and make a game that fits in this genre. The following are some of the most important game genres:

Arcade games

Here reaction speed is the most important aspect of the game. Typical examples are scrolling shooters, maze games like *Pacman*, breakout type of games, various platform games, etc. These games are relatively easy to make and normally 2-dimensional graphics is good enough for them. These are definitely the type of games you should first start creating. A particular type of arcade games is the pinball game. These are a bit harder to create because you need natural ball movement.

Puzzle games

Here clever thinking is the most important aspect. Many maze games are actually more based on puzzle solving rather than on reaction speed. Other examples include board games and sliding puzzles. These games are also normally 2-dimensional and are relatively easy to create, unless the game has to be played against a computer opponent in which case it might be difficult to program the way the computer plays the game. (Think about trying to program the computer to play chess.)

Role playing games (RPG)

Here you steer a character through a dangerous world. Typical examples are *Diablo* and *Baldur's Gate*, *Oblivion*, and *Fable*. The most important part of such a game is the development of the character you control. The character must learn new skills, become more powerful, and find additional and better weapons. At the same moment the opponents become more powerful as well. Sometimes there is also a strong storyline and the player must discover what is going on in

the world. RPG games are often isometric or fully 3D, but this is not crucial. You can also create 2-dimensional RPG games. RPG games are harder to make because you must create the mechanism of character development. Also the games normally need to be large because otherwise they are soon finished. Good level design is crucial.

Strategy games

These can be either real-time (RTS) or turn-based. Here the player normally only indirectly controls the character in the game but he does set out the strategies that the characters need to follow. Examples include *Age of Empires*, *Caesar*, *Command and Conquer*, etc. Strategy games often use an isometric view. They take a lot of time to create because they require many different game objects, like characters and buildings, that all need their own animated images and specific behavior. Moreover, they require some careful artificial intelligence for the computer opponent.

Management games

Here you must build up an empire. In these games the player manages for example a city, factory, railroad company, park, etc. Examples are *SimCity*, *Theme Park*, *Railroad Tycoon* and in some sense also games like *The Sims*. Views are often isometric for a good overview. Managing resources is a crucial ingredient. These games are difficult to make because there must be an underlying system that simulates the world, for example the behavior of the visitors of your theme park. Many GOD games can be considered as a combination of management and strategy games.

Adventure games

Here the storyline is rather crucial. Adventure games can be 2-dimensional or 3-dimensional. They typically use the well-known point-and-click interface. The difficulty in creating an adventure game does not lie in the actions but in creating an interesting, funny, and surprising story line and in creating the corresponding artwork. You really need to be an artist for this.

First-person shooters

These can be seen as the 3-dimensional version of the old arcade games. Here the emphasis is on fast-paced action and reaction speed, not on cleverness and puzzle solving. Famous examples are obviously the *Doom* and *Quake* series but huge numbers have been created. First person shooters need a 3-dimensional world to create the feeling of presence.

Third-person shooters

Here the player directly controls a game character through a hostile world. A clear example is *Tomb Raider*. The main difference with role playing games is that there is not much emphasis on character development. It is more a matter of fast action and discovering the game world. Many third-person shooters also have a storyline and borrow aspects from adventure games. Third-person shooters do not need to be 3-dimensional (think for example of the early *GTA* games) and can be created with relative ease.

Sport games

Here an existing sport, like soccer or baseball is simulated. Many such games exist and they are very popular. Creating a convincing and fun-to-play sport game is though a big challenge. It might work better if you give it a cartoon flavor because then the action does not need to be realistic.

Racing games

These are in some sense a special type of sport game. Because there are so many of them they deserve a category of their own. Some racing games, like for example many Formula-1 games, try to model the driving of a car as realistic as possible. Other games are more arcade style and make racing very easy. Racing games can be 2-dimensional or 3-dimensional. One of the major challenges when making a racing game is to create convincing racing behavior of the computer controlled opponents.

Simulators

For example flight simulators. Such games try to realistically simulate some mechanism, like a plane. They are popular because people like to understand how such systems work and like to be able to control them. Creating simulators is rather difficult because you must implement the internal working of the system you are simulating, e.g. the flying of a plane.

Clearly we did not cover all types of games in this list but it at least gives you some indication of the various genres.

You can of course produce a game that has aspects of different genres, but you should be careful with this. The player picks a game from a particular genre because he likes that genre. For example, assume that you, as a designer, decided to create an adventure game with some added action. Somewhere in the game the main character has to move to a different city and for this he has to steal a car. Chased by the police the player has to race to the next city, avoiding being caught. This may sound like fun, but be careful. A player that chooses an adventure game likes the story aspect, the fact that he has to solve complicated puzzles, and the fact that he can take his time and is not hurried. The racing part suddenly requires him to play a completely different type of game in which reaction speed counts much more than clever thinking. Probably this is not his type of game and he might be unable to finish the race and will stop playing the game. Similar problems occur for example when combining strategy games with first person shooting action. So best pick your genre and stick to it for the whole game.

Learn from Other People

This tutorial should have given you a rough idea of the things that matter when trying to create a good computer game. But in the end the best way to learn is to do it yourself and to critically look at your results.

Another piece of advice is to learn from other people's mistakes. Whenever you plan to make a particular type of game, look at similar games. Play them and see what they did right and what

they did wrong. It is amazing to see how often people repeat mistakes made by others before them.

There is a lot of information on game design available on the web and is this tutorial a lot of information was taken from these sources. You are strongly encouraged to read some of the articles experienced game designers have written. See for example the websites of Gamasutra (<http://www.gamasutra.com/>) or the Game Developers Network (<http://www.gamedev.net/>). Also many books have been written about game design although unfortunately many are rather poor.

And of course you are recommended to regularly visit our *YoYo Games* website:

<http://www.yoyogames.com/>

Here you can get help and information about how to create games, you can discuss game design issues in the forum, and you can publish your games to have them played and reviewed by others.

Further Reading

For further reading on designing games and how to create them using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

Your First Game

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: Game Maker 8.0, Lite or Pro Edition, Simple Mode

Level: Beginner

Even though *Game Maker* is very easy to use, getting the hang of it might be a bit difficult at first. This tutorial is meant for those that have some difficulty getting started with *Game Maker*. It will lead you step by step through the process of making your first game. Realize that this is the most difficult part. To make your first game you have to understand a number of the basic aspects of *Game Maker*. So please read this tutorial carefully and try to understand all the steps. Once you finished your first game the second one is going to be a lot easier.

The Game Idea

It is important that we first write a brief description of the game we are going to make. Because this is going to be our first game we better design something simple. It should keep the player interested for just a short time. Our game is going to be a little action game that we will name *Catch the Clown*. (Always try to come up with a nice name for your game.) Here is our description of the game:

Catch the Clown

Catch the Clown is a little action game. In this game a clown moves around in a playing field. The goal of the player is to catch the clown by clicking with the mouse on him. If the player progresses through the game the clown starts moving faster and it becomes more difficult to catch him. For each catch the score is raised and the goal is to get the highest possible score. Expected playing time is just a few minutes.

Clearly, a game like this will have limited appeal. But we have to start simple. Later we can add some features to the game to make it more interesting.

A Design Document

The second step in creating a game is to write a more precise design document. You are recommended to always do this before making your game, even if it is very simple. Here is our design document for *Catch the Clown*. (I omitted the description that was already given above.)

Catch the Clown Design Document

Game objects

There will be just two game objects: the clown and the wall. The wall object has a square like image. The wall surrounding the playing area is made out of these objects. The wall object does nothing. It just sits there to stop the clown from moving out of the area. The clown object has the image of a clown face. It moves with a fixed speed. Whenever it hits a wall object it bounces. When the player clicks on the clown with the mouse the score is raised with 10 points. The clown jumps to a random place and the speed is increased with a small amount.

Sounds

We will use two sounds in this game. A bounce sound that is used when the clown hits a wall, and a click sound that is used when the player manages to click with the mouse on the clown.

Controls

The only control the player has is the mouse. Clicking with the left mouse button on the clown will catch it.

Game flow

At the start of the game the score is set to 0. The room with the moving clown is shown. The game immediately begins. When the player presses the <Esc> key the game ends.

Levels

There is just one level. The difficulty of the game increases because the speed of the clown increases after each successful catch.

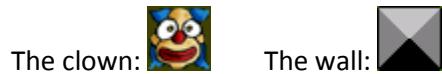
That should be good enough for the moment. We can now start creating the game. So start up *Game Maker* and let's get going. Note that this tutorial uses version 8.0 of *Game Maker*. If you use a different version, the images look a bit different. It also assumes the program runs in simple mode. You can switch between simple and advanced mode by clicking on the menu item **Advanced Mode** in the **File** menu. In advanced mode there are many more options in the different menus and forms but we won't need these for our simple game.

The game we are going to create is already given in the folder [Example](#) that comes with this tutorial. You can load it from there but you are recommended to recreate it by following the steps described below. In this way you will better understand how a game is being made in *Game Maker*. All the sprites, images, and sounds we will use are provided in the folder [Resources](#).

Adding Sprites and Sounds

As the game design document describes we will need two images for the two game objects. Such images are called sprites in *Game Maker*. There is a lot to know about sprites but for the moment, simple think

of them as little images. So we need to make or find such images. For making the images you can use any drawing program you like, for example the paint program that is part of any *Windows* system. But *Game Maker* also has a built-in drawing program for this purpose. Creating nice-looking sprites is an art that requires a lot of practice. But fortunately there are large collections of images of all sorts available for free. *Game Maker* includes a considerable number of these and on our *YoYo Games* web site (www.yoyogames.com) you can find many more. Alternatively, search the web and you are bound to find images in large quantities. For our little game we use the following two sprites, which can be found in the **Resources** folder that comes with this tutorial.



To add these sprites to the game we proceed as follows:

Creating the clown sprite resource for the game:

1. From the **Resources** menu, choose **Create Sprite**. The Sprite Properties form appears, like the one shown in Figure 1.

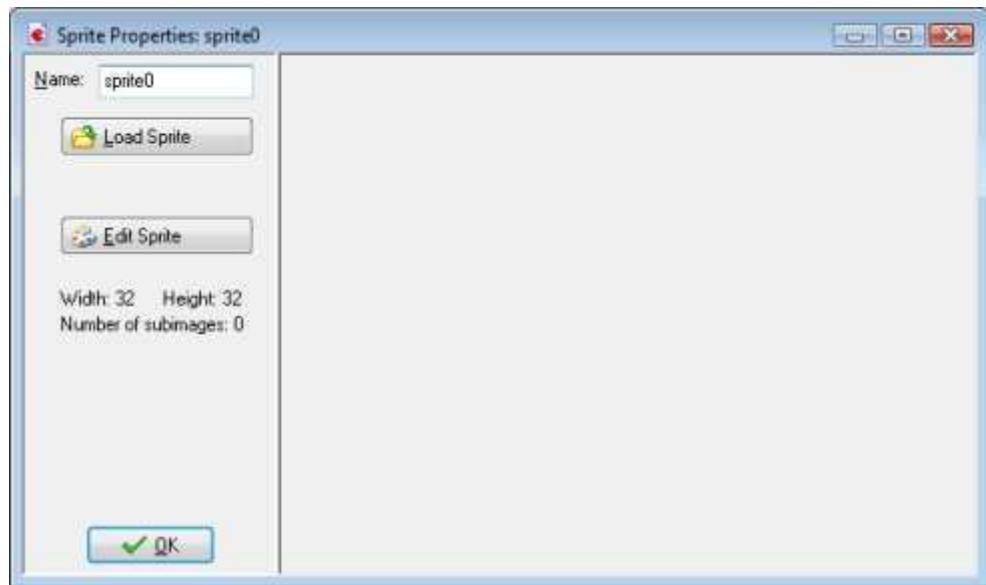


Figure 1. The empty Sprite Properties form.

2. Click on the **Name** field where currently is says `sprite0`. This is the default name for the sprite. Rename it to `spr_clown`.
3. Click on the **Load Sprite** button. This opens a file requester.
4. Navigate to the **Resources** folder that came with this tutorial and selected the image file `clown.png`. The Sprite Properties form should now look like Figure 2.
5. Press the **OK** button to close the form.

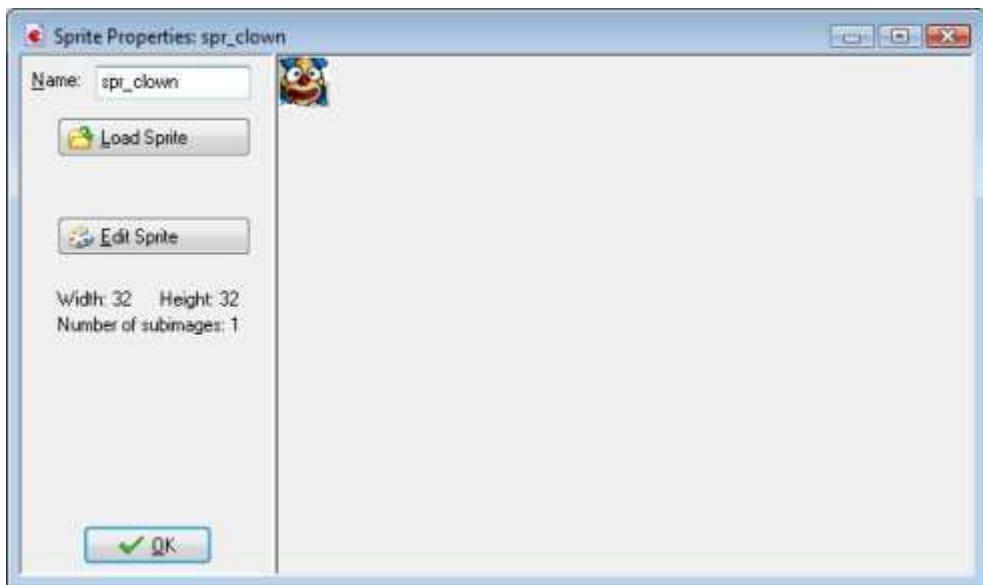


Figure 2. The clown sprite.

Next we will add the wall object in the same way.

Creating the wall sprite:

1. From the **Resources** menu, choose **Create Sprite**. Click on the **Name** field and rename it to `spr_wall`.
2. Click on the **Load Sprite** button and select the image file `wall.png`.
3. Press the **OK** button to close the form.

As you might have noticed, the clown and wall sprite have now appeared in the list of resources at the left of the *Game Maker* window. Here you will always find all the sprites, sounds, objects, rooms, etc. that you have created in your game. Together we call them the *resources* of the game. You can select a resource by clicking on its name. Now you can use the **Edit** menu to change the resource, duplicate it, or delete it. Right-clicking on the resource name will show the same menu. This overview of resources will become crucial when you are creating more complicated games.

Now that we created the sprites we will create two sound effects. One must play when the clown hits a wall and the other must play when the clown is successfully caught with the mouse. We will use two wave files for this. Wave files are excellent for short sound effects. A number of these sound effects are part of the installation of *Game Maker* and many more can be found on the web.

Create two sound resources:

1. From the **Resources** menu, choose **Create Sound**. The Sound Properties form appears. Click on the **Name** field and rename it to `snd_bounce`.
2. Click on the **Load Sound** button, navigate to the **Resources** folder that came with the tutorial, and select the sound file `bounce.wav`. The form should now look as shown in Figure 3.
3. Press the **OK** button to close the form.



Figure 3. The bounce sound resource.

4. Create another sound resource and name it `snd_click`.
5. Click the **Load Sound** button and select the sound file `click.wav`.
6. Close the form.

Within the sound properties form you can use the play button, with the green triangle pointing to the right, to listen to the sound (it is constantly repeated). Again, notice that the two sounds are shown in the list of all resources.

Objects and Actions

Having created the sprites and sounds does not mean that anything is happening. Sprites are only the images for game objects and we have not yet defined any game objects. Similar, sounds will only play if we tell them to be played. So we need to create our two game objects next.

But before we will do this you will have to understand the basic way in which *Game Maker* operates. As we have indicated before, in a game we have a number of different *game objects*. During the running of the game one or more *instances* of these game objects will be present on the screen or, more general, in the game world. Note that there can be multiple instances of the same game object. So for example, in our *Catch the Clown* game there will be a large number of instances of wall objects, which surround the playing field. There will be just one instance of the clown object.

Instances of game objects don't do anything unless you tell them how to act. You do this by indicating how the instances of the object must react to *events* that happen. There are many different events that can happen. The first important event is when the instance is created. This is the **Create Event**. Probably some action is required here. For example we must tell the instance of the clown object that it should start moving in a particular direction. Another important event happens when two instances collide with each other; a so-called **Collision Event**. For example, when the instance of the clown collides with an instance of the wall, the clown must react and change its direction of motion. Again other events happen when the player presses a key on the keyboard or clicks with mouse on an instance. For the clown we will use a **Mouse Event** to make it react to a press of the mouse on it.

To indicate what must happen in the case of an event, you specify *actions*. There are many useful actions for you to choose from. For example, there is an action that sets the instance in motion in a

particular direction, there is an action to change the score, and there is an action to play sounds. So defining a game object consists of a few aspects: we give the object a sprite as an image, we can set some properties, and we can indicate to which events instances of the object must react and what actions they must perform.

Note the distinction between *objects* and *instances* of those objects. An object defines a particular game object with its behavior (that is, reaction to events). Of this object there can be one or more instances in the game. These instances will act according to the defined behavior. Stated differently, an object is an abstract thing. Like in normal life, we can talk about a chair as an abstract object that you can sit on, but we can also talk about a particular chair, that is an instance of the chair object, which actually exists in our home.

So how does this work out for the game we are making? We will need two objects. Let us first create the very simple wall object. This object needs no behavior at all. It will not react to any events.

Create the wall object:

1. From the **Resources** menu, choose **Create object**. The Object Properties form appears, as is shown in Figure 4.

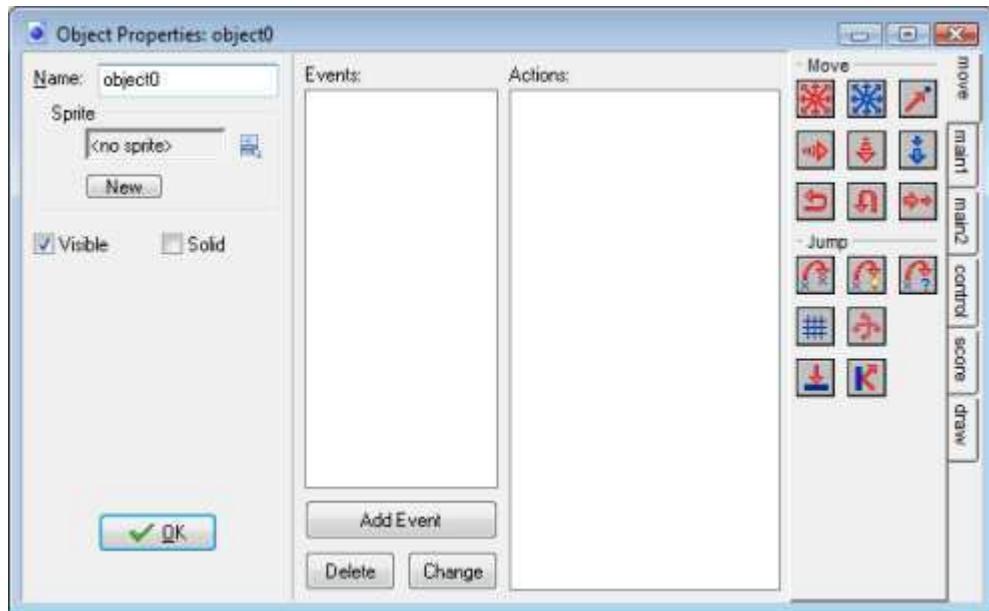


Figure 4. The empty Object Properties form.

2. Click on the **Name** field and rename the object to `obj_wall`.
3. Click on the menu icon at the end of the **Sprite** field and in the list of available sprites select the `spr_wall` sprite.
4. Instances of the wall object must be solid, that is, no other instances should be allowed to penetrate them. To this end click on the box next to the **Solid** property to enable it.
5. The filled-in form is shown in Figure 5. Press **OK** to close the form.

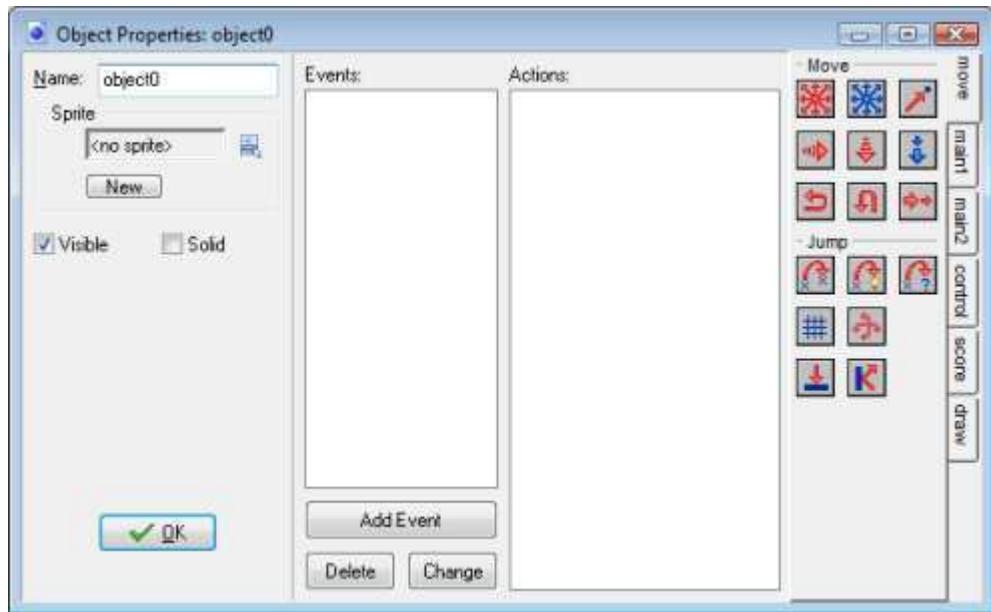


Figure 5. The filled-in properties form for the wall object.

For the clown object we start in the same way.

Create the clown object:

1. From the **Resources** menu, choose **Create object**.
2. Click on the **Name** field and rename the object to `obj_clown`.
3. Click on the icon at the end of the **Sprite** field and select the `spr_clown` sprite.

Note that we do not make the clown object solid. But for the clown there is a lot more that needs to be done. We have to specify its behavior. For this we need the rest of the form. In the middle you see an empty list with three buttons below it. This list will contain the different events that the object must respond to. With the buttons below it you can add events, delete events or change events. There are a large number of different events but you normally need just a few in your game.

Next to the events there is an empty list of actions that must be performed for the selected event (if any). And at the right of this list that are a number of tabbed pages with little icons. These icons represent the different actions. In total there are close to 100 different actions you can choose from. If you hold your mouse above one of the icons a short description of the corresponding action is given. You can drag actions from the tabbed pages at the right to the action list to make them happen when the event occurs.

We are first going to define what should happen when an instance of the clown object is created. In this case we want the clown to start moving in an arbitrary direction.

Let the clown object move:

4. Press the **Add Event** button. The Event Selector, as shown in Figure 6 will appear.



Figure 6. The Event Selector.

5. Click on the **Create** button. The create event is now added to the list of events. It is automatically selected (with a blue highlight).
6. Next you need to include a **Move Fixed** action in the list of actions. To this end, press and hold the mouse on the action image with the eight red arrows in the page at the right, drag it to the empty actions list, and release the mouse. An action form is shown asking for information about the action.
7. In the action form for the **Move Fixed** action you can indicate in which direction the instance should start moving. Select all eight directions (not the middle one; which corresponds to no motion). Note that the selected directions turn red. When multiple directions are selected one is chosen randomly. Also set the **Speed** to 4. See Figure 7 for the result. Press **OK** to indicate that we are ready with this action.

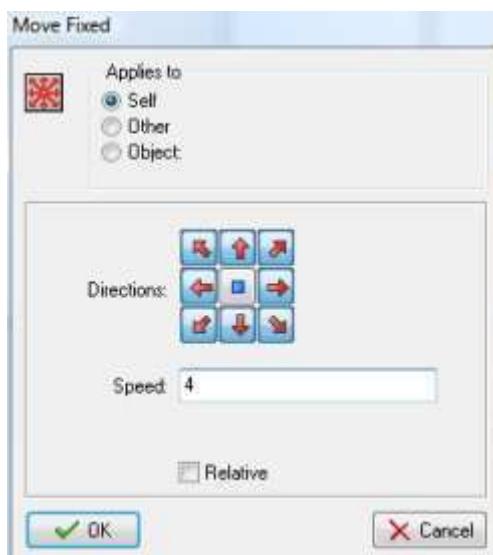


Figure 7. Setting the directions for the **Move Fixed** action.

You have now specified behavior that must be executed when an instance of the clown object is created, by adding the event, including an action, and setting the action properties. The object properties form for the clown object should now look as in Figure 8.

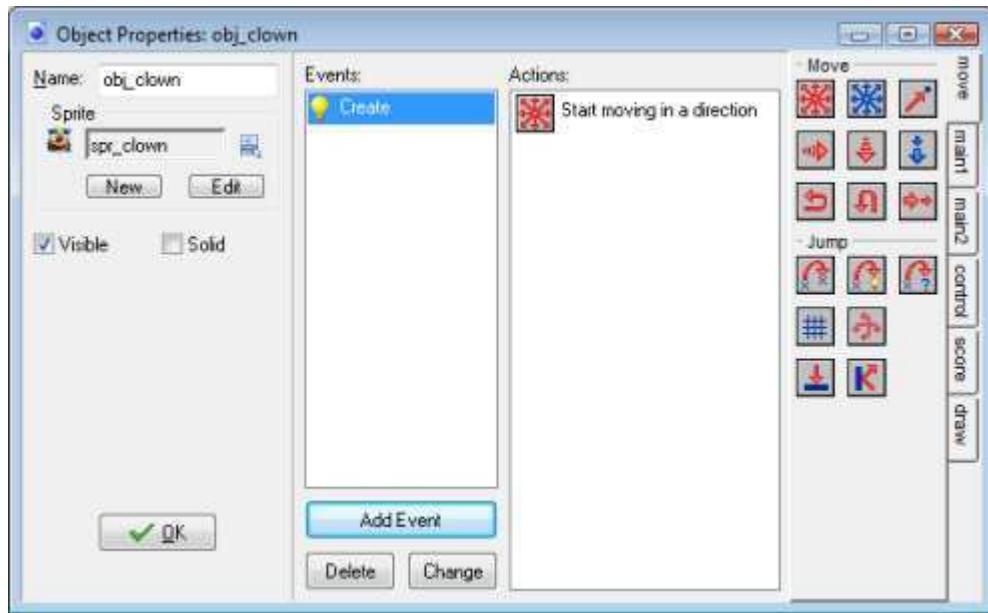


Figure 8. The properties from for the clown object after specifying the **Create** event.

The next event we will define is a collision with a wall. Here we will bounce the clown against the wall and we will play the bounce sound effect.

Handling a collision with the wall:

1. Press the **Add Event** button. In the Event Selector click on the **Collision** button and select `obj_wall`. The collision event is now added to the list of events.
2. Include a **Bounce** action by dragging it from the page at the right. The action form shown in Figure 9 will appear. There are two properties we can change but their default values are fine. We are not interested in precise bounces and we want to bounce against solid objects. (Remember that we made the wall object solid.) Press **OK** to close the action form.
3. Select the page with the tab **main1**. From it include the **Play Sound** action and drag it below the **Bounce** action already present. In the action form, click on the icon to the right of the **Sound** property and from the list select `snd_bounce`. Leave the **Loop** property to **false** as we want to play the sound only once. The form should look like in Figure 10. Press **OK** to close it.



Figure 9. The **Bounce** action form.

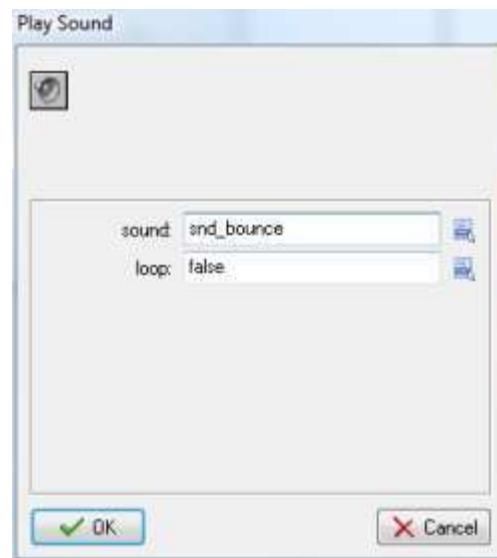


Figure 10. Playing the bounce sound effect.

That concludes the collision event with the wall object. The object properties form should now look as in Figure 11.

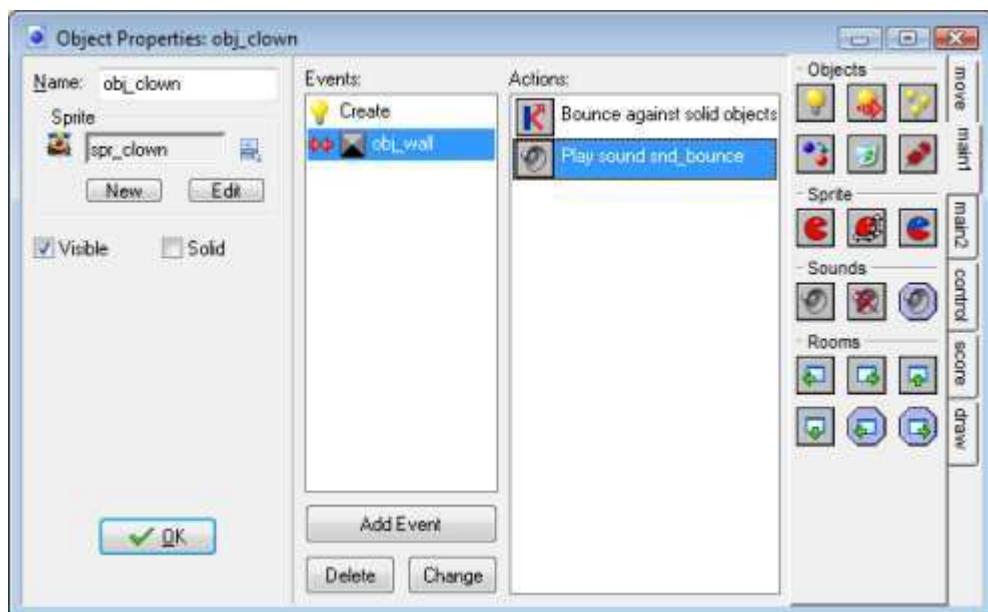


Figure 11. The collision event with the wall object.

There are two actions that are both performed (in the given order) when the collision occurs. If you for some reason made a mistake, you can right-click with the mouse on an action you added and for example choose **Delete** to remove the action (or press the <Delete> key on the keyboard). You can also choose **Edit Values** to change the properties of the action. (Double-clicking on the action will do the same.) And you can drag them up and down to change the order in which they are executed.

Finally we need to define what to do when the user clicks with the left mouse on the clown. We are going to add four actions here: First we will add 10 points to the score. This is easy as *Game Maker* automatically keeps and displays a score. Next we will play the click sound. After this we will jump the clown to a random position, and we will set a new random direction of motion with a slightly increased speed. The last two actions are added to gradually increase the difficulty of the game.

Handling a mouse press:

1. Press the **Add Event** button. In the Event Selector click on the **Mouse** button and in the menu that appears select **Left Pressed**. This event happens when the user presses the left mouse button while the mouse cursor is on top of the instance.
2. From the tabbed page labeled **score** include the **Set Score** action. As **new score** indicate a value of 10. Also click on the box next to the property **Relative** to enable it. When **Relative** is enabled the value is added to the current score. Otherwise the score would be replaced by the value. The action from should look like in Figure 12.



Figure 12. Adding 10 to the current score.

3. From the page **main1** include a **Sound** action. As **Sound** indicate `snd_click`. Leave **Loop** to false.
4. From the page **move** include a **Jump to Random** action. This action places the instance in a random collision-free position. The parameters can be left unchanged. See Figure 13.



Figure 13. Jumping to a random position.

5. Finally we include a **Move Fixed** action. Again select all eight arrows (and not the center square). As **Speed** indicate a value of 0.5 and enable the **Relative** property to add 0.5 to the current speed.

We are now ready with the clown object. We have included actions for the three events that are important. It should now look as in Figure 14. Press the **OK** button to close the form.

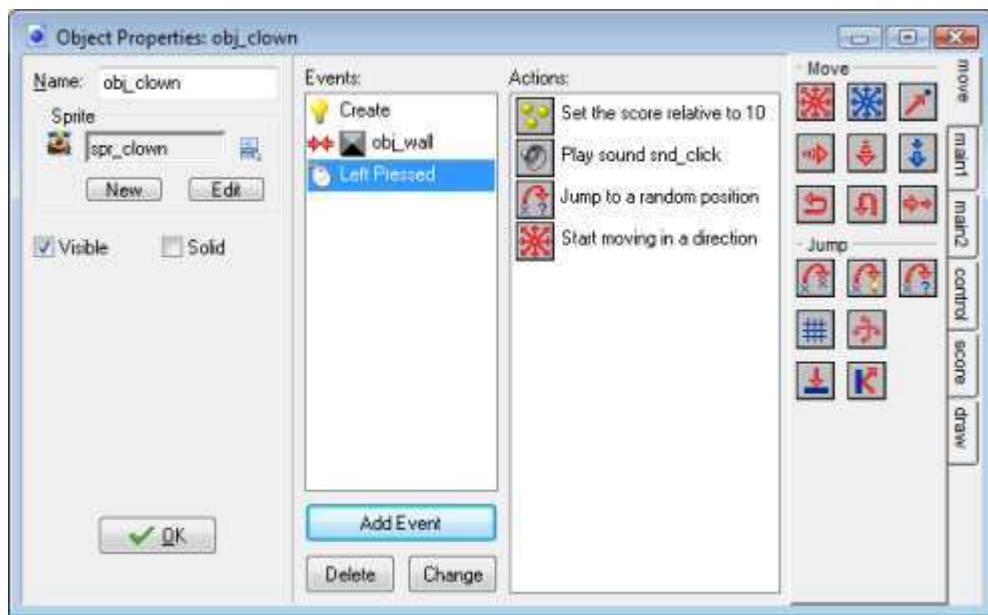


Figure 14. The clown object with all actions included.

Creating the Room

Now that we have created the game objects there is one more thing to do. We need to create the room in which the game takes place. For most games, designing effective rooms (often also called levels) is a time-consuming task because here we must find the right balance and progression in the game. But for *Catch the Clown* the room is very simple: a walled area with one instance of the clown object inside it.

Creating the room:

1. From the **Resources** menu choose **Create Room**. The Room Properties form as shown in Figure 15 will show.

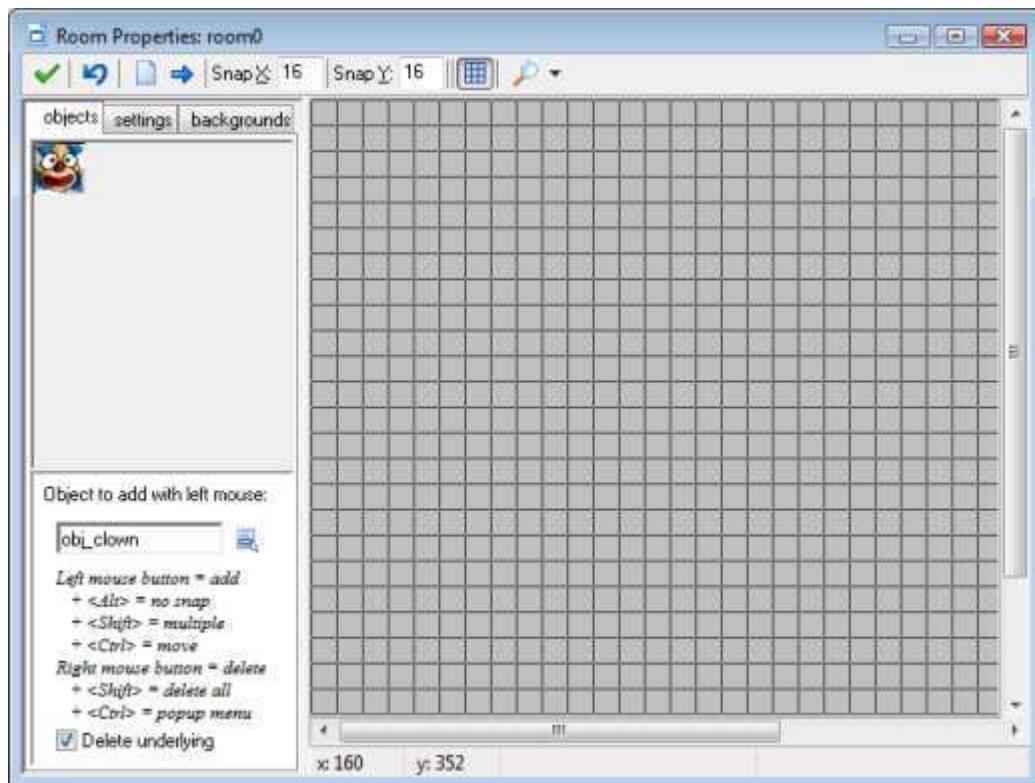


Figure 15. The Room Properties form.

2. On the left you see three tabbed pages. Select the page labeled **settings**. In the **Name** field type in `rm_main`. In the **Caption for the room** field type 'Catch the Clown'.
3. Select the **objects** tab. Enlarge the window somewhat such that you can see the complete room area at the right. At the top, change the value for **Snap X** and **Snap Y** to 32. As the size of our sprites is 32, this makes it easier to place the sprites at the correct locations.
4. At the left you see the image of the clown object. This is the currently selected object. Place one instance of it in the room by clicking with the mouse somewhere in the centre of the grey area.
5. Click on the icon with the menu symbol next to the field `obj_clown`. Here you can select which object to add. Select `obj_wall`. Click on the different cells bordering the room to put

instances there. To speed this up, press and hold the <Shift> key on the keyboard and drag the mouse with the mouse button pressed. You can remove instances using the right mouse button.

6. Press the button with the green V sign at the left top to close the form.

Saving and Testing

You might not have realized it but our game is ready now. The sprites and sounds have been added, the game objects have been designed and the first (and only) room in which the game takes place has been created. Now it is time to save the game and to test it.

Saving the games works as in almost any other Windows program. Choose the command **Save** from the **File** menu, select a location and type in a name. *Game Maker* games get a file extension **.gmk**. Note that you cannot directly play such game files. You can only load them in *Game Maker*. Below we will see how to make stand-alone game executables.

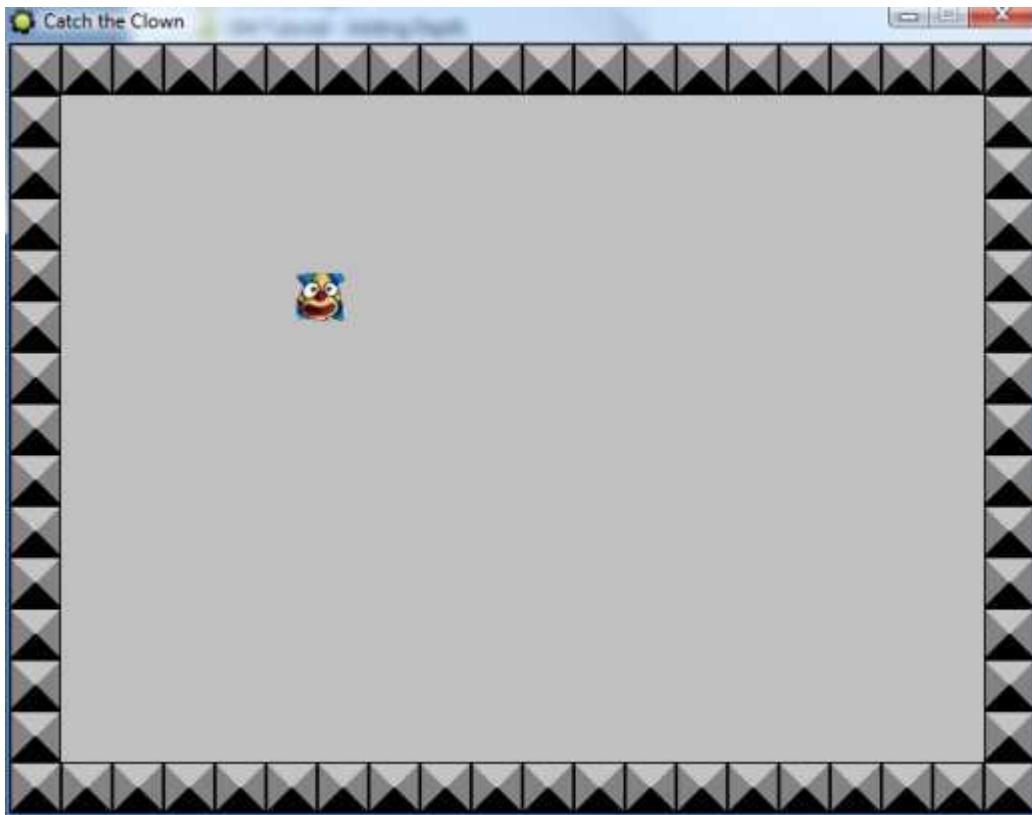


Figure 16. Playing the game.

Next we need to test the game. Testing is crucial. You can test it yourself but you should also ask others to test it. Testing (or running the game in general) is simple; choose the command **Run normally** from the **Run** menu. The design window will disappear, the game will be loaded and, if you did not make any mistakes, the room will appear on the screen with the clown moving inside it, as in Figure 16. Try clicking on it and see whether the game behaves as expected. You should hear the correct sounds and the speed

of the clown should increase. To end the game, press the <Esc> key or click on the close button at the top right of the window. The design window will reappear.

Now it is time to fine tune the game. You should ask yourself for example the following questions: Is the initial speed correct? Is the increase in speed correct? Is the room size correct? Did we pick effective sprites and sounds for the game? If you are not happy, change these aspects in the game and test again. Remember that you should also let somebody else test the game. Because you designed the game it might be easier for you than for other people.

Once you are happy with your game you should create a stand-alone executable for the game. This is a version of the game that can run without the need for *Game Maker*. This is very simple. In the **File** menu choose the command **Create Executable**. You have to indicate the place and name of the stand-alone executable and you are done. You can now close *Game Maker*, and run the executable game. You can also give the executable to your friends and let them play it, or you can publish it on the YoYo Games website <http://www.yoyogames.com> for people to download. (Of course we do not recommend you to place this exact copy of the *Catch the Clown* game there. Better create your own original game.) For this you will need some screenshots of the game, which you can easily make by pressing <F9> while running the game.

Finishing touches

Our first game is ready but it needs some finishing touches to make it a bit nicer. First of all we are going to add some background music. This is very easy.

Create background music:

1. From the **Resources** menu, choose **Create Sound**. The Sound Properties form appears. Click on the **Name** field and rename it to `snd_music`.
2. Click on the **Load Sound** button, navigate to the **Resources** folder and select the sound file `music.mid`. Note that this is a midi file. Midi files are often used for background music as they use less memory.
3. Press the **OK** button to close the form.
4. Reopen the clown object by double clicking on it in the resource list at the left of the window.
5. Select the **Create** event. From the **main1** page include a **Play Sound** action. As **Sound** indicate `snd_music` and set **Loop** to true because we want the music to repeat itself forever.

Secondly we are going to add a background image. The grey background of the room is rather boring. To this end we use a new type of resource, the background resource.

Add a background image:

1. From the **Resources** menu, choose **Create Background**. The Background Properties form appears. Click on the **Name** field and rename it to `back_main`.
2. Click on the **Load Background** button, navigate to the **Resources** folder and select the image file `background.png`. The form should now look like Figure 17.

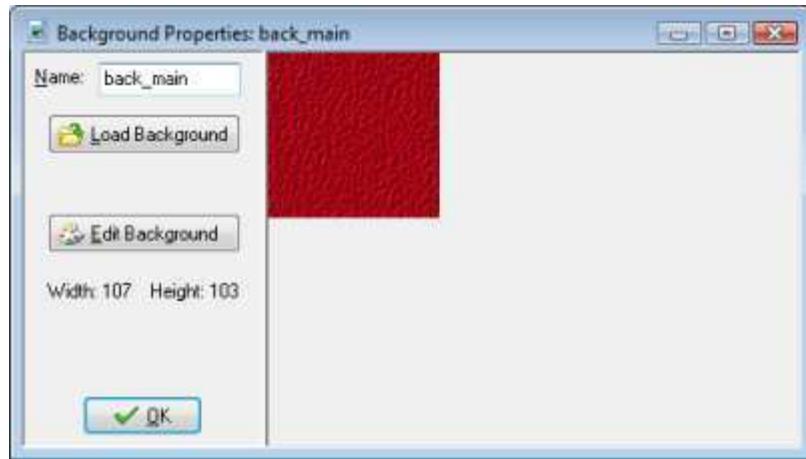


Figure 17. The Background Properties form.

3. Press **OK** to close the form.
4. Reopen the room by double clicking on it in the resource list.
5. Select the **backgrounds** tab. Deselect the property **Draw background color**.
6. Click on the little menu icon in the middle and pick the `back_main` in the popup menu. As you will see, in the room we suddenly have a nice background, As in Figure 18. Note the two properties **Tile Hor.** and **Tile Vert.**. They indicate that the background must be tiled horizontally and vertically, that is, repeated to fill the whole room. For this to work correctly the background image must be made such that it nicely fits against itself without showing cracks.

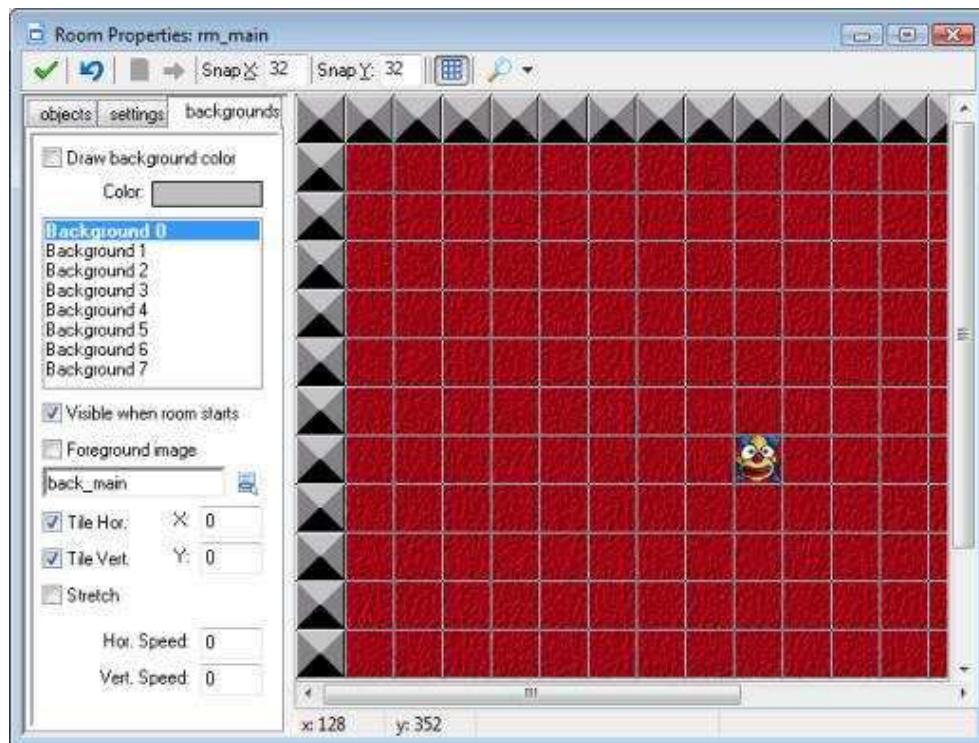


Figure 18. The room with a background.

If you play the game a bit you will see that it is very easy because you know exactly where the clown is going. To make it more difficult we let the clown change its direction of motion from time to time. To this end we are going to use an alarm clock. Each instance can have multiple alarm clocks. These clocks tick down and at the moment they reach 0 an **Alarm** event happens. In the creation event of the clown we will set the alarm clock. And in the alarm event we change the direction of motion and set the alarm again.

Adding the alarm clock:

1. Reopen the clown object by double clicking on it in the resource list at the left of the window.
2. Select the **Create** event. From the **main2** page include a **Set Alarm** action. As **Number of steps** indicate 50. The alarm number we keep as Alarm 0. See Figure 19.



Figure 19. Setting the alarm clock to 50 steps.

3. Click on **Add Event**. Choose the button **Alarm** and in the popup menu select **Alarm 0**.
4. In the event include the **Move Fixed** action (from the **move** tab). Select all eight arrows. Set the **Speed** to 0 and enable the **Relative** property. In this way 0 is added to the speed, that is, it does not change.
5. To set the alarm clock again, include a **Set Alarm** action. As **Number of steps** again indicate 50.



We set the alarm clock to 50 steps but you might wonder what a step is. Default *Game Maker* takes 30 steps per second. So 50 steps is slightly more than 1.5 seconds. (You can change the game speed in the **settings** tab in the room properties.)

Finally, each game must tell the players what the goal is and how the user plays the game. So some help is required. *Game Maker* has a standard mechanism for this.

Adding a help text:

1. From the **Resources** menu select **Change Game Information**. A simple text editor will appear.

2. Type in some useful information for the player, in particular about the goal of the game and the way to control it. You can use different fonts, sizes, and colors. See for example Figure 20.

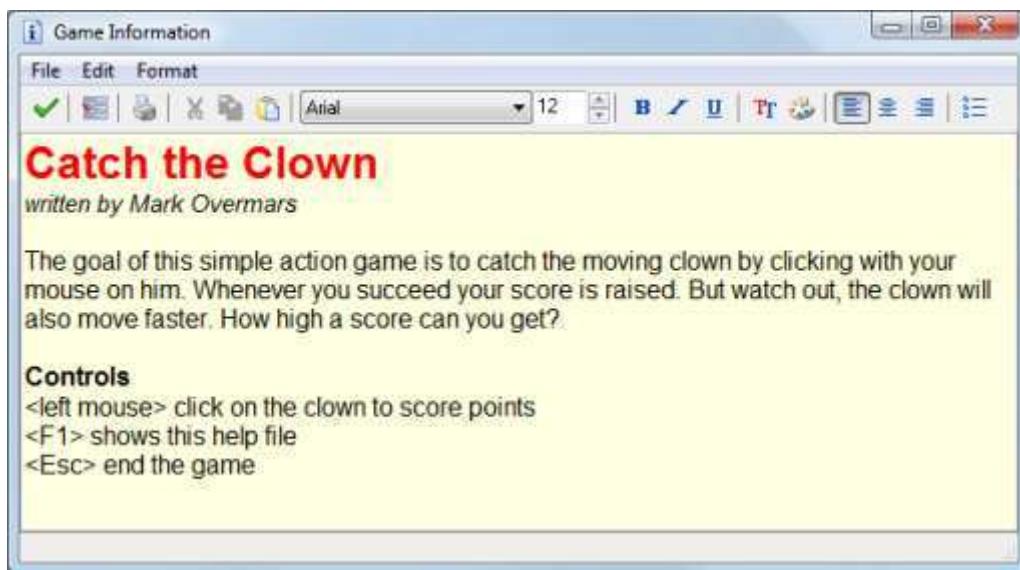


Figure 20. Adding some help for the player.

During the game this text is automatically shown when the player presses the F1 key (like in most other programs).

Your First Game is Ready

Congratulations. You finished your first game. And the first game is always the most difficult one. You also learned about the most important aspects of *Game Maker*: sprites, images and sounds, the game objects, events and actions, and the rooms.

Before continuing with a new game you might want to play a bit more with the *Catch the Clown* game. Here are some things you might want to try to add:

- Have two clowns moving around. (This is extremely easy because you can place multiple instances of the same object in a room.)
- Have a different dark clown that you should not catch because it will cost you part of your score.

But you surely can come up with other creative ideas.

In this tutorial we have only covered some of the most basic aspects of *Game Maker*. We discussed only a few events and actions. There are many more for you to explore. You can try to do so yourself or you can download and read one of the other tutorials which you can download from

<http://www.yoyogames.com>

Further Reading

For further reading on creating games using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

The book gives a step-by-step introduction into many aspects of *Game Maker* and in the process you create nine beautiful games that are also fun to play.

Creating Maze Games

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd

Last changed: December 23, 2009

Uses: Game Maker 8.0, Lite or Pro Edition, Advanced Mode

Level: Beginner

Maze games are a very popular type of game and they are easy to create in *Game Maker*. This tutorial shows in a number of easy-to-follow steps how to create such a game. The nice part is that from the first step on we have a playable game that, in further steps, becomes more extended and more appealing. All partial games are provided in the folder [Examples](#) that comes with this tutorial and can be loaded into *Game Maker*. Also all resources are provided in the folder [Resources](#).

The Game Idea

Before starting creating a game we have to come up with an idea of what the game is going to be. This is the most important (and in some sense most difficult) step in designing a game. A good game is exciting, surprising and addictive. There should be clear goals for the player, and the user interface should be intuitive.

The game we are going to make is a maze game. Each room consists of a maze. To escape the maze the player must collect all diamonds and then reach the exit. To do so the player must solve puzzles and monsters must be avoided. Many puzzles can be created: blocks must be pushed in holes; parts of the room can be blown away using bombs, etc. It is very important to not show all these things in the first room. Gradually new items and monsters should appear to keep the game interesting.

So the main object in the game is a person controlled by the player. There are walls (maybe different types to make the maze look more appealing). There are diamonds to collect. There are items that lie around that do something when picked up or touched by the player. One particular item will be the exit of the room. And there are monsters that move by themselves. But let us tackle these things one by one.

A Simple Start

As a first start we forget about the diamonds. We will create a game in which the player simply must reach the exit. There are three crucial ingredients in the game: the player, the wall, and the exit. We will need a sprite for each of them and make an object for each of them. You can find the first very simple game under the name [maze_1.gmk](#). Please load it and check it out.

The objects

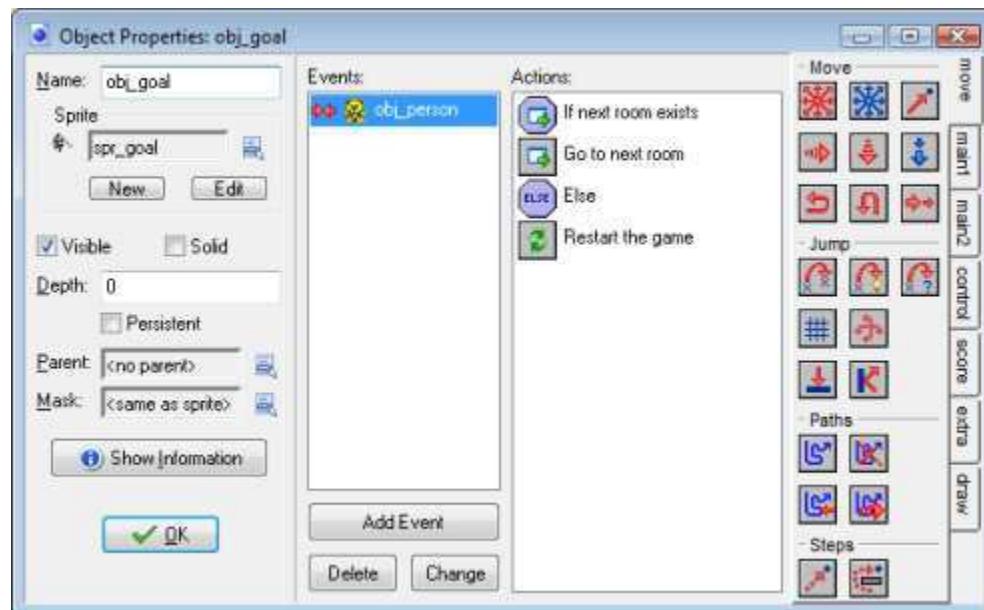
Let us first create the objects. For each of the three objects we use a simple 32x32 sprite:



Create these three sprites in the usual way and name them `spr_person`, `spr_wall`, and `spr_goal`.

Next we create three objects. Let us first make the wall object. We will give it the `spr_wall` sprite as image, name it `obj_wall` and make it solid by checking the box labeled **Solid**. This will make it impossible for other objects, in particular the person, to penetrate the wall. The wall object does not do anything else. So no events need to be defined for it.

Secondly, let us create the goal object. This is the object the player has to reach. It is a non-solid object. We decided to give it a picture of a finish flag. This makes it clear for the player that he has to go here. When the person collides with it we need to go to the next room. So we put this action in this collision event (it can be found in the tab **main1**). This has one drawback. It causes an error when the player has finished the last room. So we have to do some more work. We first check whether there is a further room. If so we move there. Otherwise we restart the game. So the event will look as follows:



Obviously, in the full game we better do something more when the player finishes the last level, like showing some nice image, or giving him a position in the list of best players. We will consider this later.

Finally we need to create the person that is controlled by the player. Some more work is required here. It must react to input from the user and it should not collide with a wall. We will use the arrow keys for movement. (This is natural, so easy for the player.) There are different ways in which we can make a

person move. The easiest way is to move the player one cell in the indicated direction when the player pushed the arrow key. A second way, which we will use, is that the person moves in a direction as long as the key is pressed. Another approach is to keep the player moving until another key is pressed (like in PacMan).

We need actions for all four arrow keys. The actions are rather trivial. They simply set the right direction of motion. (As speed we use 4.) To stop when the player releases the key we use the keyboard event for <no key>. Here we stop the motion. There is one complication though. We really want to keep the person aligned with the cells of the grid that forms the maze. Otherwise motion becomes rather difficult. E.g. you would have to stop at exactly the right position to move into a corridor. This can be achieved as follows. In the **control** tab there is an action to test whether the object instance is aligned with a grid. Only if this is the case the next action is executed. We add it to each arrow key event and set the parameters to 32 because that is the grid size in our maze:



Clearly we also need to stop the motion when we hit a wall. So in the collision event for the person with the wall we put an action that stops the motion. There is one thing you have to be careful about here. If your person's sprite does not completely fill the cell, which is normally the case, it might happen that your character is not aligned with a grid cell when it collides with the wall. (To be precise, this happens when there is a border of size larger than the speed around the sprite image.) In this case the person will get stuck because it won't react to the keys (because it is not aligned with the grid) but it can also not move further (because the wall is there). The solution is to either make the sprite larger, or to switch off precise collision checking and as bounding box indicate the full image.

Creating rooms

That was all we had to do in the actions. Now let us create some rooms. Create one or two rooms that look like a maze. In each room place the goal object at the destination and place the person object at the starting position.

Done

And that is all. The first game is ready. Play a bit with it. E.g. change the speed of the person in its creation event, create some more levels, change the images, etc.

Collecting Diamonds

But the goal of our game was to collect diamonds. The diamonds itself are easy. But how do we make sure the player cannot exit the room when not all diamonds are collected? To this end we add a door object. The door object will behave like a wall as long as there are still diamonds left, and will disappear when all diamonds have gone. You can find the second game under the name `maze_2.gmk`. Please load it and check it out.

Beside the wall, goal, and person object we need two more objects, with corresponding sprites: the diamond and the door. The diamond is an extremely simple object. The only action it needs is that it is destroyed when the person collides with it. So in the collision event we put an action to delete it. The door object will be placed at a crucial place to block the passage to the goal. It will be solid (to block the person from passing it). In the collision event of the person with the door we must stop the motion. In the step event of the door we check whether the number of diamonds is 0 and, if so, destroys itself. There is an action for this. We will also play some sound such that the player will hear that the door is opened. So the step event looks as follows:



Making it a bit nicer

Now that the basics of the game are in place, let us make it a bit nicer.

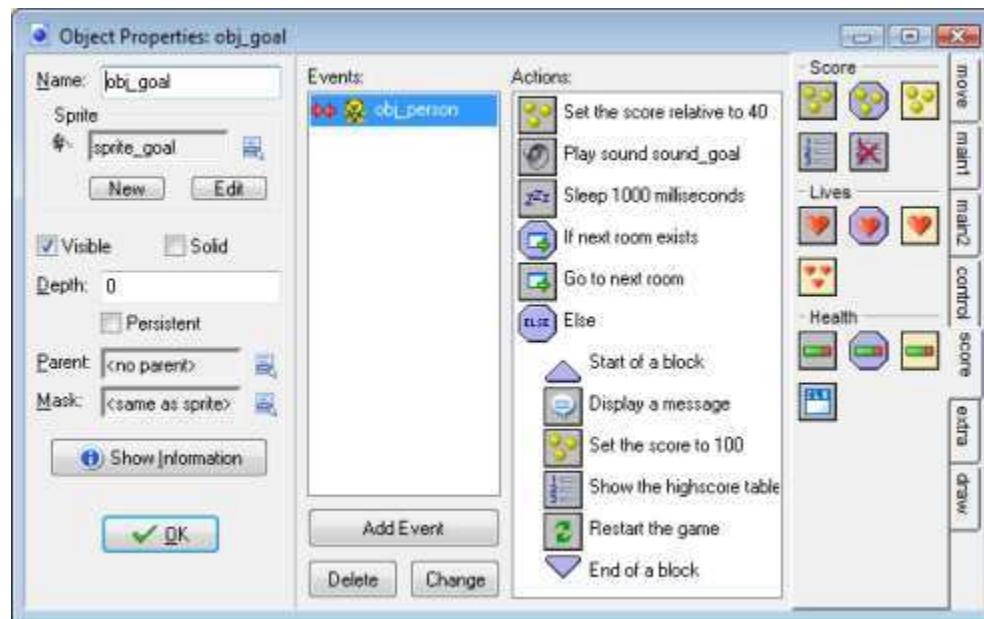
The walls look pretty ugly. So let us instead make three wall objects, one for the corner, one for the vertical walls, and one for the horizontal walls. Give them the right sprites and make them solid. Now with a bit of adaptation of the rooms it looks a lot nicer. Giving the rooms a background image also helps.

To avoid having to specify collision events of the person with all these different walls (and later similar for monsters), we use an important technique in *Game Maker*. We make the corner wall object the parent of the other wall objects. This means that the wall objects behave as special variants of the corner wall object. So they have exactly the same behavior (unless we specify different behavior for them). Also, for other instances, they are the same. So we only have to specify collisions with the corner. This will automatically be used for the other wall objects. Also the door object we can give as parent the corner wall.

Score

Let us give the player a score such that he can measure his progress. This is rather trivial. For each diamond destroyed we give 5 points. So in the destroy event of the diamond we add 5 points to the score. Finishing a level gives 40 points so we add 40 points to the score in the collision event for the goal with the person.

When the player reaches the last room a high-score table must be shown. This is easy in *Game Maker* because there is an action for this. The goal object does become a bit more complicated though. When it collides with the person the following event is executed:



It adds something to the score, plays a sound, waits a while and then either goes to the next room, if this is the last room, shows a message, the high-score table, and restarts the game.

Note that the score is automatically displayed in the caption. This is though a bit ugly. Instead we will create a controller object. It does not need a sprite. This object will be placed in all rooms. It does some

global control of what is happening. For the moment we just use it to display the score. In its drawing event we set the font and color and then use the action to draw the score.

Starting screen

It is nice to start with a screen that shows the name of the game. For this we use the first room. We create a background resource with a nice picture. (You might want to indicate that no video memory should be used as it is only used in the first room.) This background we use for the first room (best disable the drawing of the background color and make it non-tiled.) A start controller object (invisible of course) is created that simply waits until the user presses a key and then moves to the next room. (The start controller also sets the score to 0 and makes sure that the score is not shown in the caption.)

Sounds

A game without sounds is pretty boring. So we need some sounds. First of all we need background music. For this we use some nice midi file. We start this piece of music in the `start_controller`, looping it forever. Next we need some sound effects for picking up a diamond, for the door to open, and for reaching the goal. These sounds are called in the appropriate events described above. When reaching the goal, after the goal sound, it is good to put a little sleep action to have a bit of a delay before going to the next room.

Creating rooms

Now we can create some rooms with diamonds. Note that the first maze room without diamonds we can simply leave in. This is good because it first introduces the player to the notion of moving to the flag, before it has to deal with collecting diamonds. By giving a suggestive name to the second room with the diamonds, the player will understand what to do.

Monsters and Other Challenges

The game as it stands now starts looking nice, but is still completely trivial, and hence boring to play. So we need to add some action in the form of monsters. Also we will add some bombs and movable blocks and holes. The full game can be found in the file `move_3.gmk`.

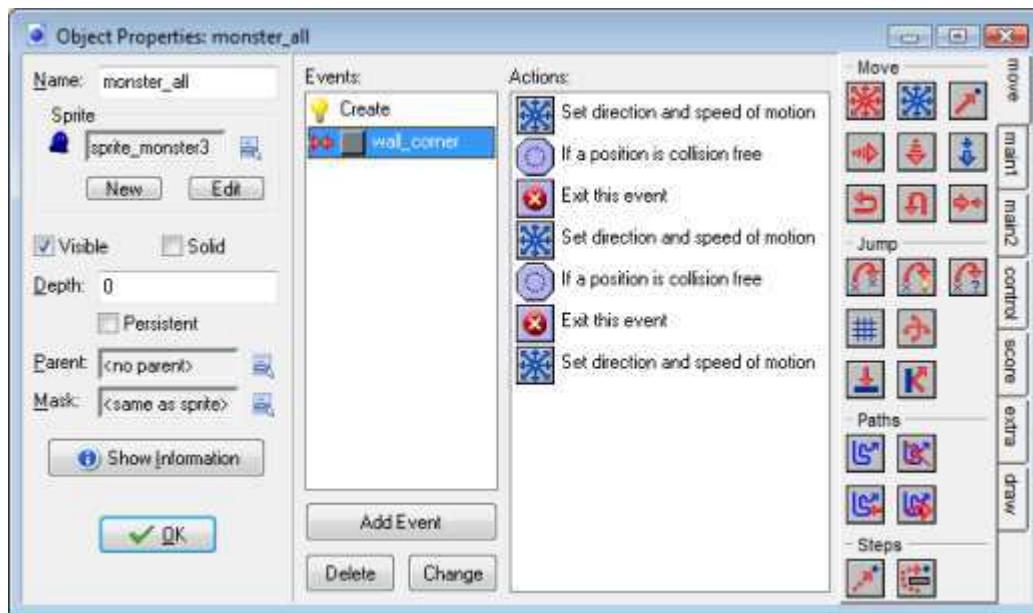
Monsters

We will create three different monsters: one that moves left and right, one that moves up and down, and one that moves in four directions. Adding a monster is actually very simple. It is an object that starts moving and changes its direction whenever it hits a wall. When the person hits a monster, it is killed, that is, the level is restarted and the player loses a life. We will give the person three lives to start with.

Let us first create the monster that moves left and right. We use a simple sprite for it and next create an object with the corresponding sprite. In its creation event it decides to go either left or right. Also, to make life a bit harder, we set the speed slightly higher. When a collision occurs it reverses its horizontal direction.

The second monster works exactly the same way but this time we start moving either up or down and, when we hit a wall, we reverse the vertical direction.

The third monster is slightly more complicated. It starts moving either in a horizontal or in a vertical direction. When it hits a wall it looks whether it can make a left or a right turn. If both fail it reverses its direction. This looks as follows:

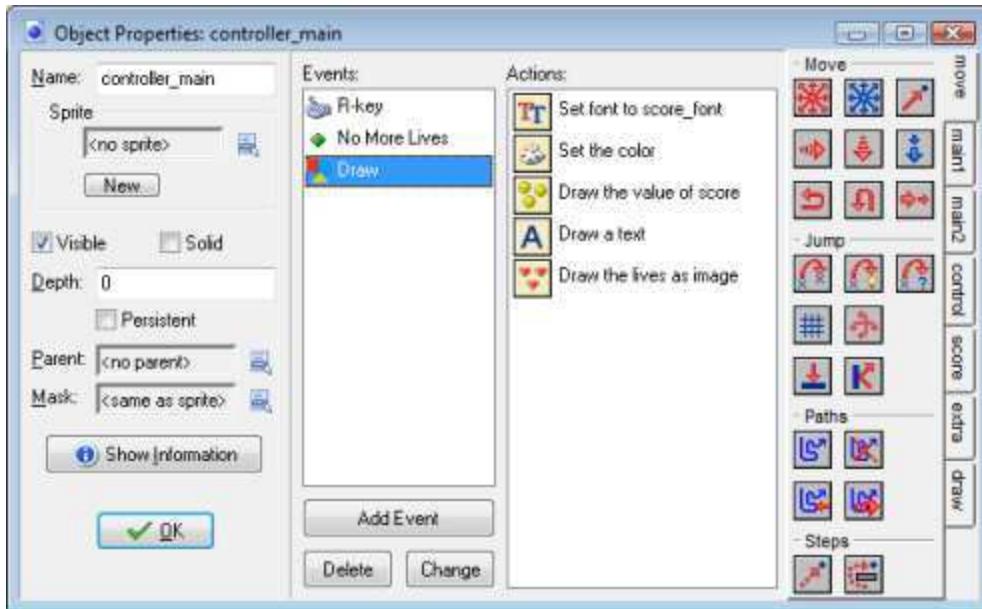


To avoid problems with monsters being slightly too small, we uncheck precise collision checking and modify the mask to set the bounding box to the full image.

When the person collides with a monster, we have to make some awful sound, sleep a while, decrease the number of lives by one, and then restart the room. (Note that this order is crucial. Once we restart the room, the further actions are no longer executed.) The controller object, in the "no more lives" event, shows the high-score list, and restarts the game.

Lives

We used the lives mechanism of *Game Maker* to give the player three lives. It might though be nice to also show the number of lives. The controller object can do this in the same way as with the score. But it is nicer if you actually see small images of the person as lives. There is an action for this in the **score** tab. The drawing event now looks as follows:



Note that we also used a font resource to display the score.

Bombs

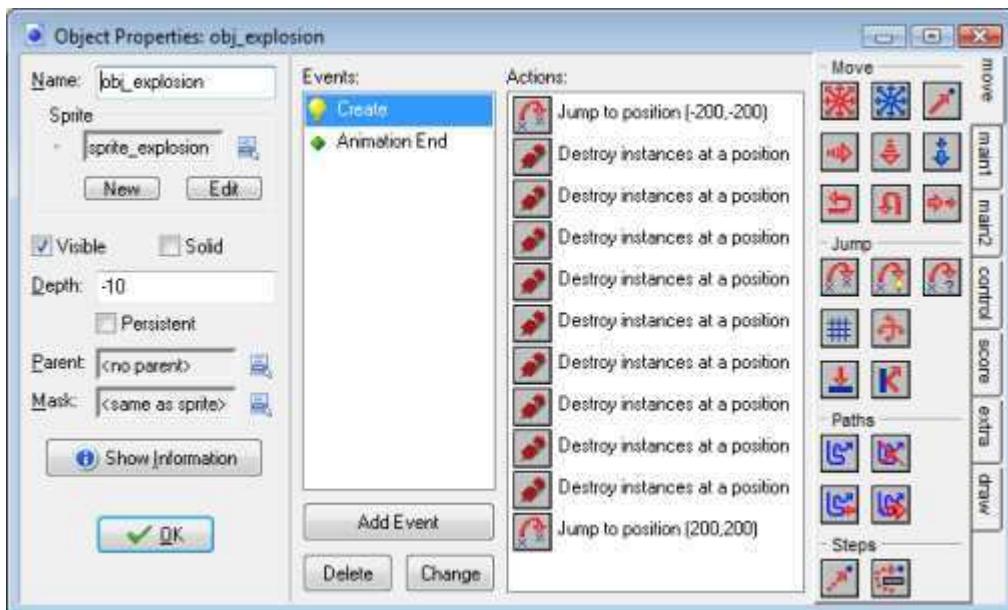
Let us add bombs and triggers to blow them up. The idea is that when the player gets to the trigger, all bombs explode, destroying everything in their neighborhood. This can be used to create holes in walls and to destroy monsters. We will need three new objects: a trigger, a bomb, and an explosion. For each we need an appropriate sprite.

The bomb is extremely simple. It just sits there and does nothing. To make sure monsters move over it (rather than under it) we set its depth to 10. Object instances are drawn in order of depth. The ones with the highest depth are drawn first. So they will lie behind instances with a smaller depth. By setting the depth of the bomb to 10 the other objects, that have a default depth of 0, are drawn on top of it.

The trigger is also rather simple. When it collides with the person it turns all bombs into explosions. This can be achieved by using the action to change an object in another object. At the top we indicate that it should apply to all bombs.



The explosion object just shows the animation. After the animation it destroys itself. (You have to be careful that the origin of the explosion is at the right place when turning a bomb into it.) The object also must destroy everything it touches. This requires a little bit of work. First of all, we do not want the explosion to destroy itself so we move it temporarily out of the way. Then we use actions to destroy all instances at positions around the old position of explosion. Finally we place the explosion back at the original place.



Note that this goes wrong if the person is next to the bomb! So make sure the triggers are not next to the bombs. It is important to carefully design the levels with the bombs and triggers, such that they present interesting challenges.

Blocks and holes

Let us create something else that will enable us to make more complicated puzzles. We create blocks that can be pushed by the player. Also we make holes that the player cannot cross but that can be filled with the blocks to create new passages. This allows for many possibilities. Blocks have to be pushed in a particular way to create passages. And you can catch monsters using the blocks.

The block is a solid object. This main problem is that it has to follow the movement of the person when it is pushed. When it collides with the person we take the following actions: We test whether relative position `8*other.hspeed, 8*other.vspeed` is empty. This is the position the block would be pushed to. If it is empty we move the block there. We do the same when there is a hole object at that position. To avoid monsters running over blocks we make the corner wall the parent of the block. This does though introduce a slight problem. Because a collision event is defined between the person and the corner wall and not between the person and the block, that event is executed, stopping the person. This is not what we want. To solve this we put a dummy action (just a comment) in the collision event of the person with the block. Now this event is executed instead, which does not stop the person. (To be precise, the new collision event overrides the collision event of the parent. As indicated before, you can use this to give child objects slightly different behavior than their parents.)

The hole is a solid object. When it collides with the block it destroys itself and the block. We also make the corner wall its parent to let it behave like a wall.

With the blocks and holes you can create many intriguing rooms. There is though a little problem. You can easily lock yourself up such that the room can no longer be solved. So we need to give the player the possibility to restart the level, at the cost of one life. To this end we use the key R for restart. In this keyboard event for the controller we simply subtract one from the lives and restart the room.

This finishes our third version of the maze game. It now has all the ingredients to make a lot of interesting levels.

Some Final Improvements

Let us now finalize our game. We definitely should improve the graphics. Also we need a lot more interesting levels. To do this we add some bonuses and add a few more features. The final game can be found in the file `maze_4.gmk`.

Better graphics

The graphics of our current game is rather poor. So let us do some work to improve it. The major thing we want to change is to make the person look in the direction he is going. The easiest way to achieve this is to use a new image that consists of 4 subimages, one for each direction, as follows:



Normally *Game Maker* cycles through these subimages. We can avoid this by setting the variable `image_speed` to 0. When we change the direction of the character, we can change the subimage shown by the action to change the sprite:



A similar thing we can do for all the monsters but here there are no explicit events where we change the direction. It is easier to add a test in the end step event to see in which direction the monster is moving and adapt the sprite based on that.

Bonuses

Let us add two bonuses: one to give you 100 points and the other to give you an extra life. They are both extremely simple. When they meet the person they play a sound, they destroy themselves, and they either add some points to the score or 1 to the number of lives. That is all.

One way streets

To make the levels more complicated, let us add one-way streets that can only be passed in one direction. To this end we make four objects, each in the form of an arrow, pointing in the directions of motion. When the person is completely on it we should move it in the right direction. We do this in the step event of the person. We check whether the person is aligned to the grid in the right way and whether it meets the particular arrow. If so, we set the motion in the right direction. (We set it to a speed of 8 to make it more interesting.)

Frightened monsters

To be able to create Pacman like levels we give every monster a variable called `afraid`. In the creation event we set it to 0 (false). When the person meets a new ring object we set the variable to true for all monsters and we change the sprite to show that the monster is indeed afraid. Now when the person meets the monster we first check whether it is afraid or not. If it is afraid the monster is moved to its initial position. Otherwise, the player loses a life. See the game for details.

Now let's make a game out of it

We now have created a lot of object, but we still don't have a real game. A very important issue in games is the design of the levels. They should go from easy to hard. In the beginning only a few objects should be used. Later on more objects should appear. Make sure to keep some surprises that only pop up in level 50 or so. Clearly the levels should be adapted to the intended players. For children you definitely need other puzzles than for adults.

Also a game needs documentation. In *Game Maker* you can easily add documentation using the **Game Information**. Finally, players won't play the game in one go. So you need to add a mechanism to load and save games. Fortunately, this is very easy. *Game Maker* has a built-in load and save mechanism. F5 saves the current game, while F6 loads the last saved game. You should though put this in the documentation.

You find a more complete game, including all this in the file [maze_4.gmk](#). Please load it, play it, check it out, and change it as much as you like. In particular, you should add many more levels (there are only 20 at the moment). Also you can add some other objects, like e.g. keys that open certain doors, transporters that move you from one place in the maze to another, bullets that the person can shoot to kill monsters, doors that open and close from time to time, ice on which the person keeps moving in the same directions, shooting traps, etc.

Finally

I hope this tutorial helped you in creating your own games in *Game Maker*. Remember to first plan your game and then create it step by step (or better, object by object). There are always many different ways in which things can be achieved. So if something does not work, try something else. Good luck!

Further Reading

For further reading on creating games using *Game Maker* you are recommended to buy our book:

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

The book gives a step-by-step introduction into many aspects of *Game Maker* and in the process you create nine beautiful games that are also fun to play.