PORTFOLIO

Dustin Gibson
334-492-0050
dustingibson@outlook.com
http://dustingibson.com

Contents

Major Projects

Source code and executable file for each project are found in:
http://bit.ly/dustingibson

C# - TaskBuddy

Motivation and Description:
Often times users work offline and lack a fast searching dictionary or a decent spell checking tool. Even if the user is online, spelling one word on a program lacking a spellcheck feature can be a frustrating experience as the user must clear his working space, pull a word processor or browser to copy and paste the correct spelling of a single word. The TaskBuddy opens the program with one global shortcut without requiring the user to launch any large programs or obscure the working area. In addition, the word selected is automatically copied and pasted upon a single click. This program is designed to reduce unnecessary steps to accomplish multiple tasks.

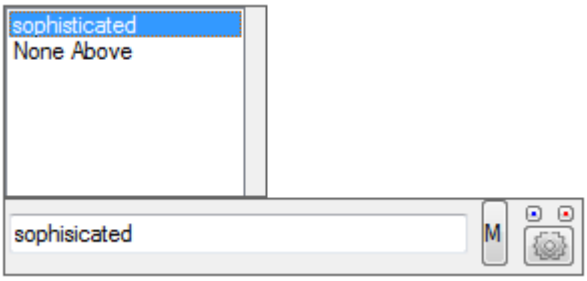
Technical Experience Gained:
1. Minimization UI design
2. Multiple uses of dictionaries and lists
3. Use of Visual Studio debugging tools
4. DLL imports
5. Spell check algorithm
6. Text and file processing

Details:

UI

To insure portability, the titlebar and framing is removed. Two small appropriately colored buttons are included to allow the user to close and minimize the application. The user is given an option to include a taskbar button, but by default it's included. The program is designed to store relative screen position upon exiting. When a user drags the application to a new location and restarts the application, it will appear at that new location. In an addition, the application will always appear on top with the exception of the taskbar. Users can immediately show the application through the global hotkey, “Windows key + A”.

Spell Check Algorithm

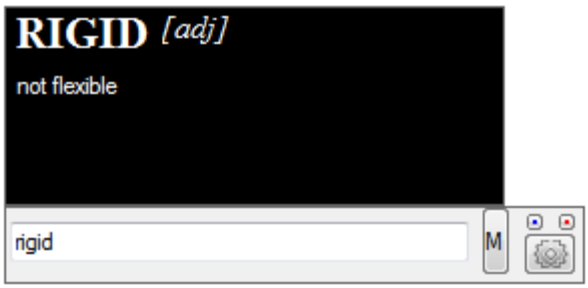


The algorithm used in this program is the same algorithm described in Peter Norving's paper http://norvig.com/spell-correct.html). In summary, the algorithm follows a series of operations to form new strings using the testing string as input. These new strings are checked from the dictionary and if the entry exist then it's added to a list of suggestions. There are five different routines that together provides a set of suggest words. First the testing string is partitioned into the i^{th} index and each character that follows. For an example: the word "lamp" is partitioned in four parts (null string + lamp, l + amp, la + mp, lam + p) These routines are:

1. Delete – A character is deleted at each position of the test string.
2. Transpose – The partitioned list is rearranged such that the 2nd index appears before the 1st index.
3. Insert – This routine involves adding each letter of the alphabet in between the splits. Using the letter p and the test string "wenesday" on iteration would look like "wenespday".
4. Replace – Same functionality as the insert routine with the exception that the first character in the 2nd partition of each split is replaced by each character of the alphabet.

There are ways to improve accuracy by following the same set of procures, but for two characters instead of one. For an example, it will be more accurate if the insertion procedure inserts each character of the alphabet and a permutation of two letters of the alphabet. This will significantly hinder the speed because of the dramatic increase of iterations in the insertion and replacement routines.

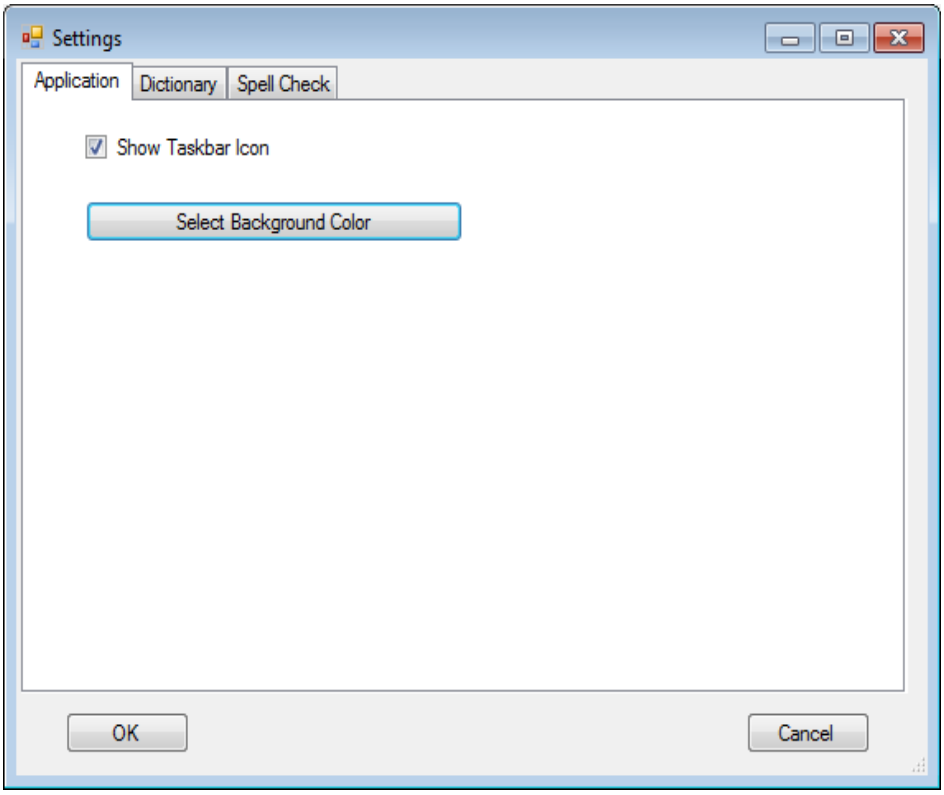
Dictionary



The dictionary feature of the program is simple. The application takes advantage of GNU's Aspell dictionary text file, which includes valuable information such as the word, definition, and parts of

speech. Each of which are separated with a delimiter. The program isolates each part and record the information to the appropriate data structure.

Settings



1. Hide/Show taskbar icon
2. Choose background color for the application
3. Choose background and text color for dictionary display
4. Allow auto copy and paste for selected corrections

Choosing ok saves the settings to a textfile and restarts the application with the new loaded settings.

Java – Graph Theory Demonstration

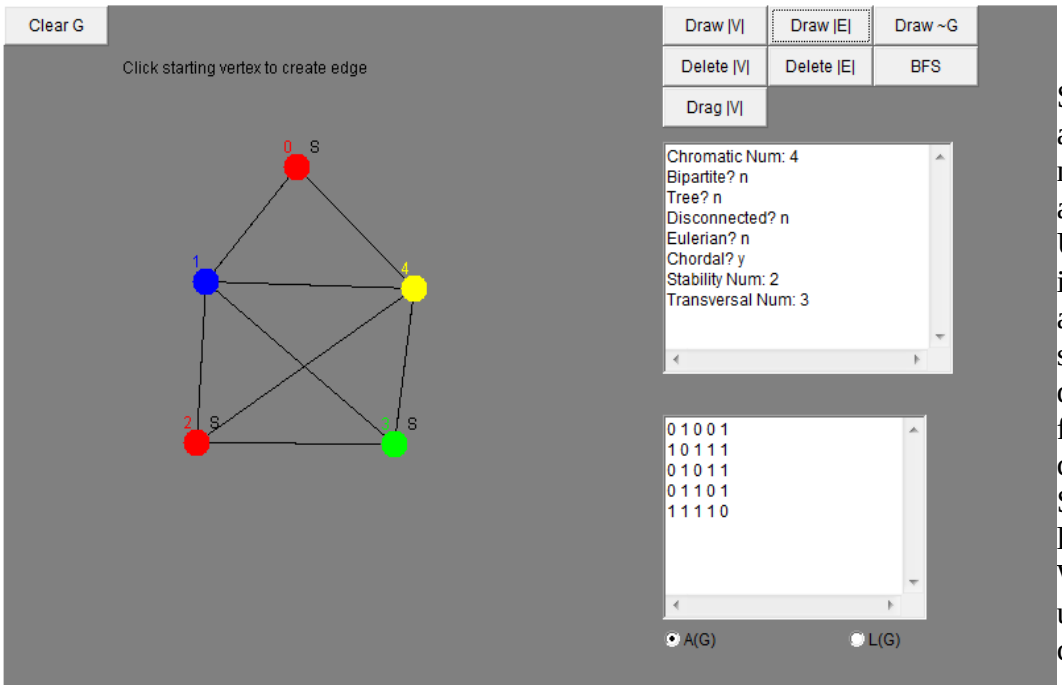
Description and Motivation:
I completed this program shortly before the completion of Algorithmic Graph Theory course. The purpose of the program is to give information to particular graph drawn by the user. The information includes colors, chromatic number, type of graph (chordal, tree, etc), breadth search path, the matrix data structure, and much more. The user selects the vertices and places it inside of the applet. Then the user can connect two vertices by clicking the edge button, clicking on the starting vertex, and then the terminal vertex. The user also can delete vertices and edges. The information displayed on the right will automatically update after the user performs drawing tasks.

Technical Experience Gain:
1. Java applet UI (similar to Swing)
2. Java web applet development
3. Implementation of dozens of Graph Algorithms
4. Uses of stacks and queues

References:
Dr. Voloshin
Email: vvoloshin@troy.edu
Phone: 334-670-3552
Website: http://spectrum.troy.edu/voloshin/
Title: Troy University Mathematics Professor

Details:

UI



Since this application is a Java applet, the UI implementation is slightly different from the classical Swing library. When a user creates an

edge, the edge changes to a pink color and updates a line from the cursor to the first selected vertex.

Data Structure

The data structure for the graph drawn by the user is a simple adjacency matrix. Essentially, it's a two dimensional vector. There is a slight tradeoff between speed, space, and simplicity. The space needed for the adjacency matrix is the number of nodes squared. Even though the space of an adjacency list is much less than the adjacency matrix (only the number of nodes), adjacency matrix is mathematically superior since algebraic operations are more piratical to matrices than lists.

Graph Coloring

The graph coloring algorithm uses a greedy approach to color the nodes so that no corresponding node share the same color.

Let N_m be a list of nodes in a graph with m nodes. Let i=0 and col_list be a list of colors recorded:

STEP 1: Examine nodes adjacent to N_i . Let's call a list of these adjacent nodes A. Let j=0.

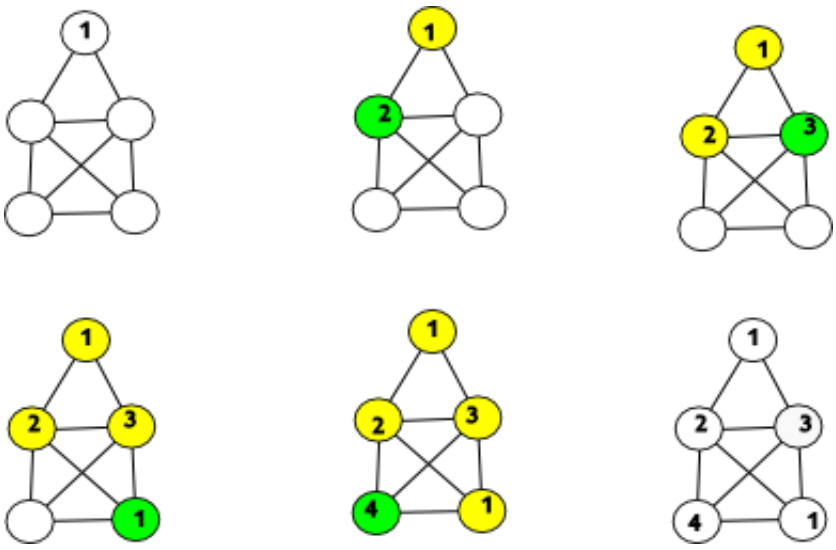
STEP 2: Eliminate any colors from col_list that appear on each node adjacent to A_j.

STEP 3: If all colors are eliminated from the list of colors. Increase the list by one and add a new color. Color A_j with the new color. If there is a color available on the list, color the node with that color.

STEP 4: Increase j by 1. Repeat step 2 until there are no adjacent nodes left.

STEP 5: Check to see if every node is colored. If not then increase i by one. If so end the algorithm.

Note in the example below, numbers are used in place of actual color.



A complete graph is the worst case scenario since each node is connected and the algorithm requires a search for each adjacent node. The first iteration will only take n steps where n is the number of nodes. Since the first iteration is already colored, it can be ignored. Therefore the second iteration takes n-1 steps. Repeating the pattern, the third iteration takes only n-2 steps. The pattern is a triangular sum.

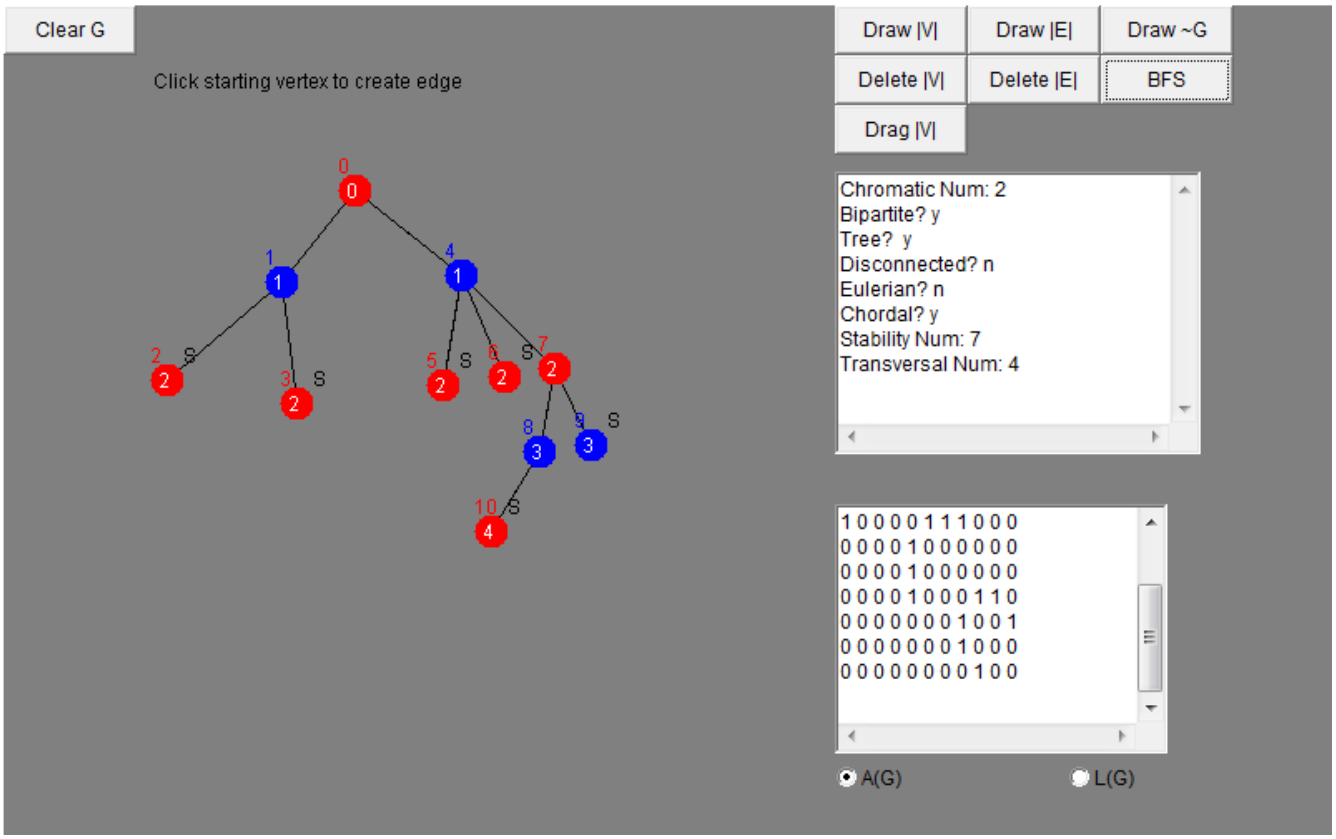
$$n+(n-1)+(n-2)+...$$
$$\sum_{i=0}^{n} i=\frac{n(n+1)}{n}\in O(n^2)$$

The best case scenario is a graph where each node has a degree of 1. This is simply O(1).

The number listed on the program as “chromatic number” is simply the number of colors.

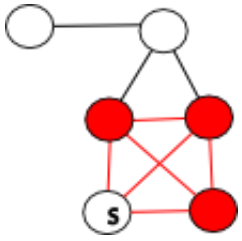
Breadth Search

An iterative form of the breadth first search algorithm is used for the bfs feature of this application. Instead of a list of nodes appeared in order of appearance in the bfs algorithm, the degree of separation as figured through the bfs algorithm is much more useful and applicable.

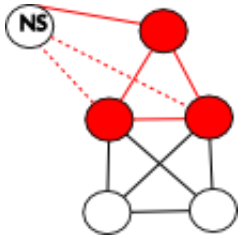


Simplicial vertices

A simplicial vertex is a special kind of vertex, mainly essential for chordal graphs operations. To tell if a particular vertex is simplicial, observe each adjacent vertices as a subgraph. If the graph is a complete graph (all nodes connected to all of the other remaining nodes) then the deleted vertex is simplicial, otherwise it is not simplicial. The implementation is trivial, create a copy of the current graph, retrieve the adjacency matrix of only the adjacent nodes, and analyze if the subgraph is complete. A matrix of a complete graph will follow a certain pattern; a diagonal of zeros will appear from topmost left of the matrix to topmost right.



Notice how each node adjacent to the marked node are completely connected to each other.



This example is not simplicial since one of the nodes adjacent to the marked nodes are disconnected to one of the other adjacent nodes. This connection is displayed as a dashed edge where the edge would have been to make the marked vertex simplicial.

Chordal Graph Identification

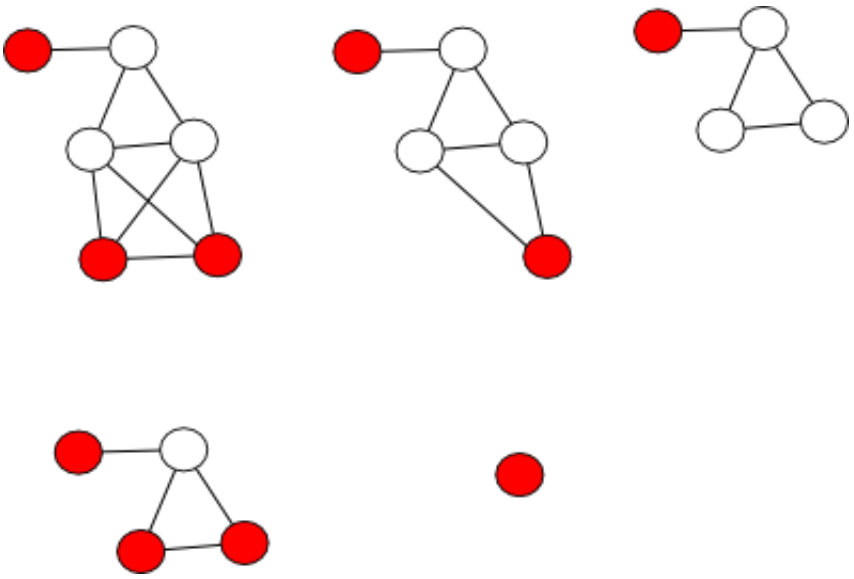
The program successfully attempts to determine if a given graph is chordal. A chordal graph is a special type graph with a wide range of application where each cycle of size more than four contains a chord (geometrically speaking, a line crossing from one end of a structure to another). I have used a popular algorithm that takes advantage of simplicial vertices to identify if a graph is chordal.

STEP 1: Find each simplicial vertices. The program does this automatically whenever a user draws a graph.

STEP 2: Delete a simplicial vertex to obtain a subgraph.

STEP 3: Repeat step 1 with the resulting subgraph until the graph is empty or if there are no simplicial vertices.

STEP 4: If the graph is empty then the original graph is a chordal graph. If the graph contains no simplicial vertices and it's not empty then the original graph is not chordal.



Tree Identification

A tree is another type of graph that does not have cycles. Since cycles have the same number of edges as do nodes, the number of nodes trees will always have is one node more than edges. Many graphs can be eliminated as trees by noticing that this fact is false. However, there will be many other graphs that follow the formula, but are not trees.

Let N be a list of nodes. i = 0

STEP 1: Check to see if the particular graph follows this formula: e = n-1 where e is the number of edges and n is the number of nodes. If it does not follow then it is not a tree. If it does then continue with step 2.

STEP 2: Obtain a list of nodes adjacent to node N_i. Let's call this list A. Traverse each node listed on A. If the node traversed is node N_i, return false. Otherwise repeat step 2 for i=i+1. If all nodes are visited (i > n), return true.

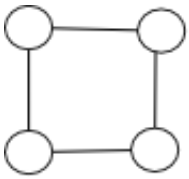
Other Trivial Features

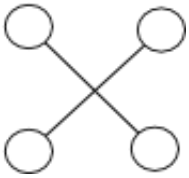
Bipartite Graph – Since all bipartite graph is colored using two colors, the program checks if a graph has two colors.

Disconnected Graph – Identifying a disconnected graph just requires a quick examination of the adjacency matrix. If any row contains only zeros then the graph is disconnected.

Eulerian Graph – The degrees of a graph is updated each time a user updates the graph. An eulerian graph is a type of graph where one is able to trace the graph without visiting each node twice. All eulerian graph has an even node. A simple checking using mod 2 on each node is used to determine if

the given graph is eulerian.





Complementary Graph – If the user selects “Draw ~G” a complementary graph is drawn. The adjacency matrix is inverted to create a complementary graph.

Java – Randomness Tester

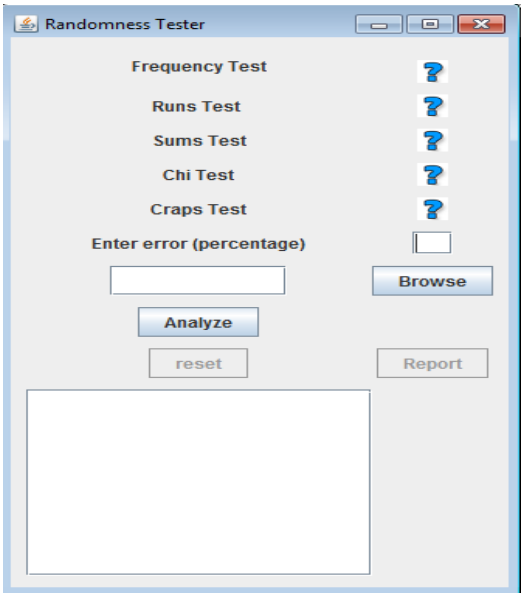
Description and Motivation:
Student Support Services' math specialist and also my former supervisor when I worked as a tutor, Andrew Williamson, was searching for a programmer to complete his randomness radio project. Mr. Williamson modified a radio from Radio Shack to produce white noise. He wanted to use to noise to generate a series of random numbers. Without any luck, Williamson could not find any easy to use randomness tests. I volunteered to develop a program that allow him to easily test his results. The tester is simple to use, import any file, click analyze, and if the user desires a report, he or she may wish to click report to generate the detailed result of each test in .pdf format. The results includes graphs and a black and white image representing the bit by bit file information.

Technical Experience Gain:
1. Java UI development using Swing
2. Chart and .PDF creation using third party libraries
3. Advance uses of Java file streaming library
4. Randomness testing algorithms
5. Heavy research in statistics and probability
6. Working with a team of one person

Reference:
Andrew Williamson
Email: aewilliamson@troy.edu
Phone: 334-670-5985
Title: Troy University Math Specialist (Student Support Services)

Details:

UI



The library used from the randomness tester is the standard java swing library. The goal of the layout is to be as minimalistic as possible and display more detailed information in a separate report as a pdf file. The user should be able to use the program multiple times in one session, so a reset feature is added. The textbox below is selected results from the runs test.

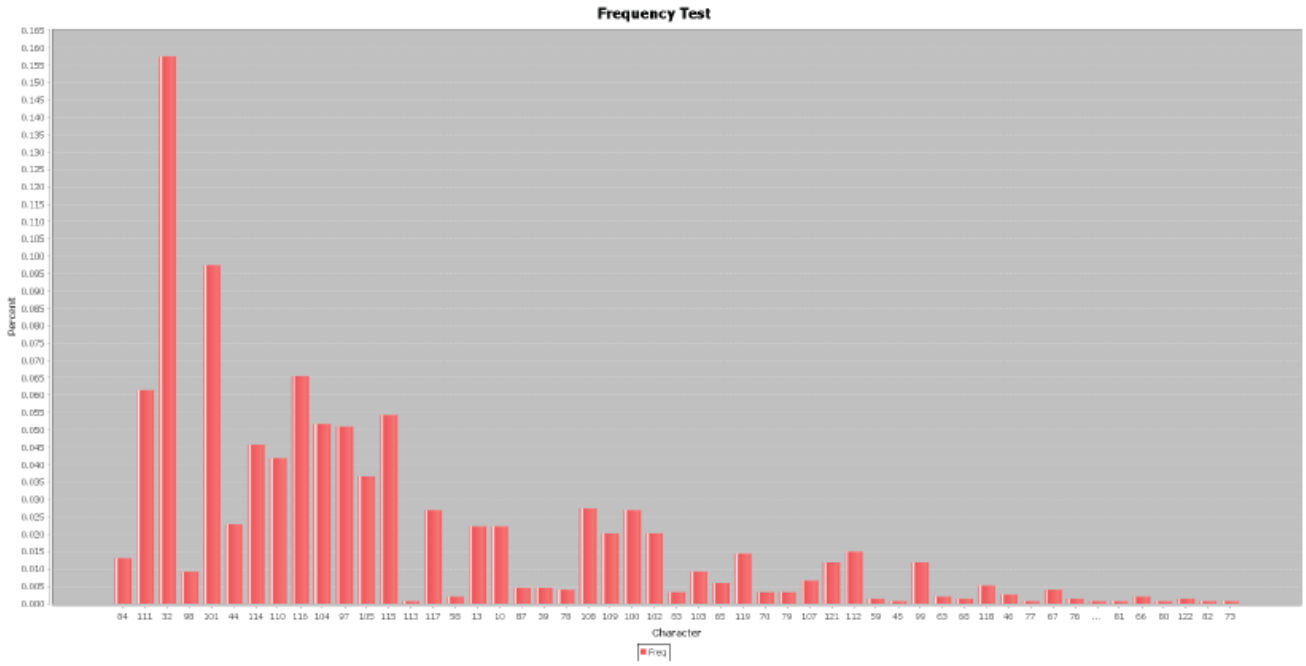
Concept

Determining whether one set of data is random can be described as subjective. A set of data that is random lacks pattern and structure. However, a problem arises when one attempts to define these terms because there are infinite number of patterns. The best way to tell if a set of data is random is to run a series of test and let the user decide from these test if the set is random. This particular problem uses five tests, each serving a different purpose. Since the user ultimately decides randomness based on the resulting data, the user must also input the margin of error as a percentage. A 1% margin of error will hav moree precise results than a 15% margin of error.

Data Structure

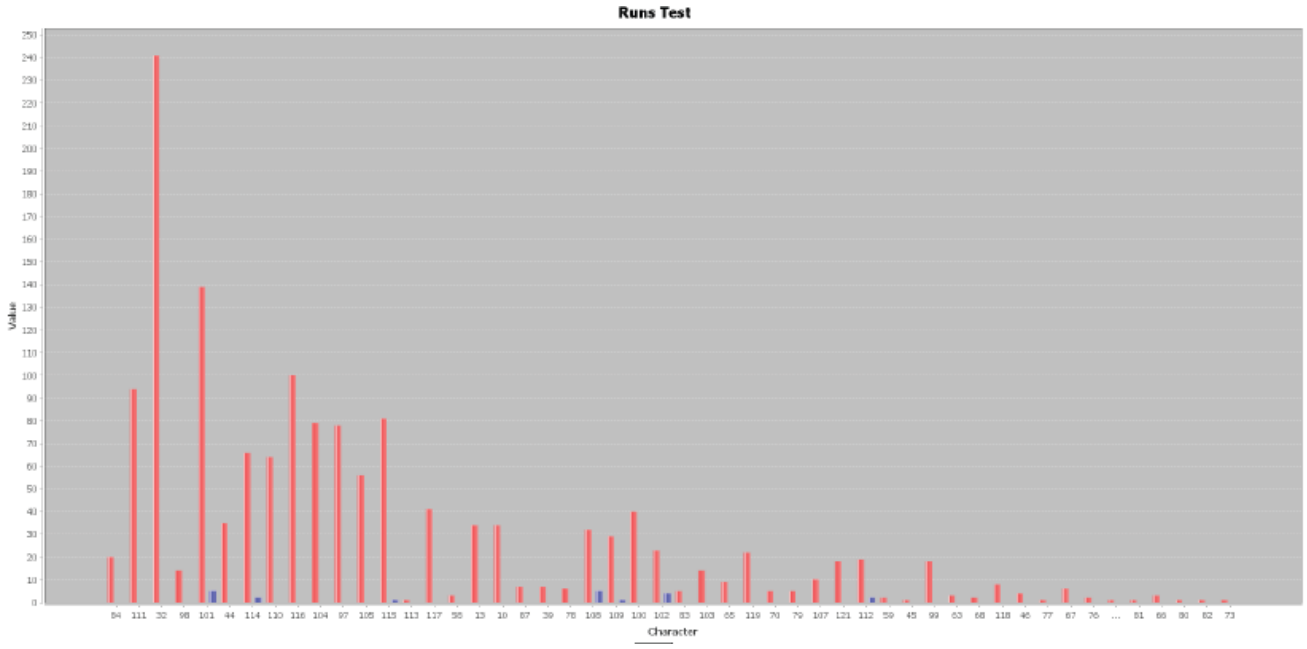
Majority of users will upload text files. It's important to track each character, the number of times they appear in the file, and number of continuous runs. When the user runs after uploading a file, the program scans the document for unqiue character and then it's recorded to an “alphabet” object. Afterwards, the program counts the number of each alphabet appears and that number is inserted as a value into a hash table with the alphabet being the key. The number of runs of a particular character is also inserted into a hash table

Frequency Test



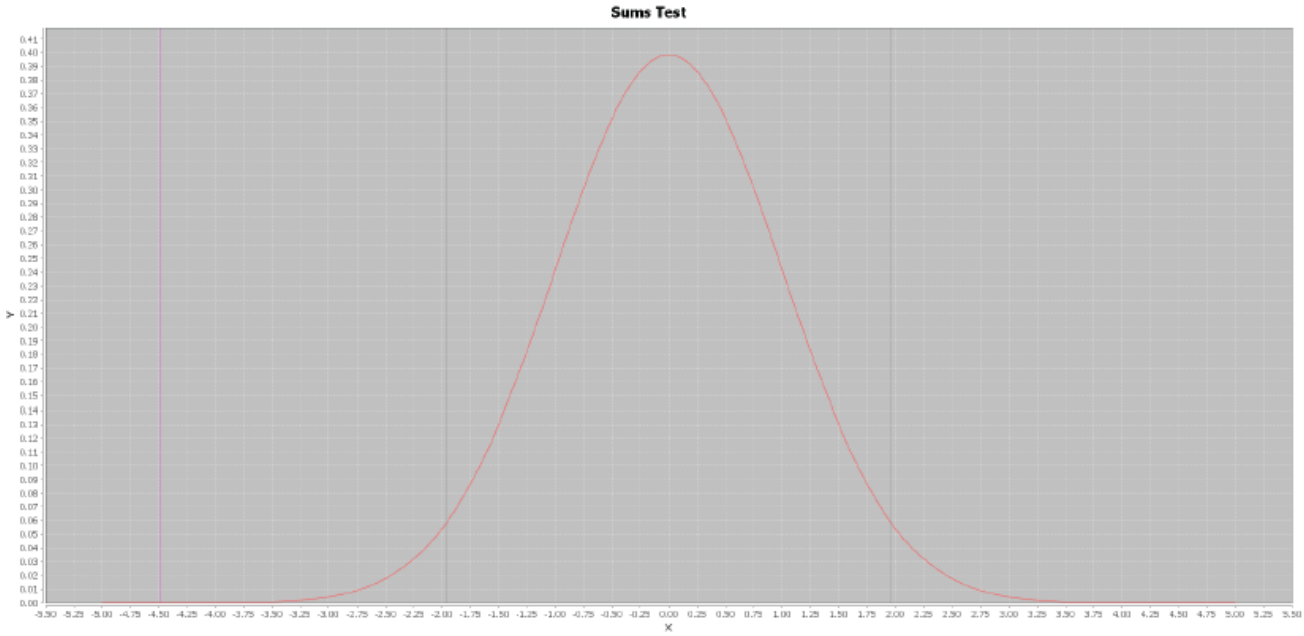
The most simplistic randomness test is the frequency test. Each character in the program's character must have a uniform distribution within a plus or minus margin of error. Suppose our test document has three letters (a, b, and c) and suppose the total amount of characters in our document is 30. A perfect uniform distribution means that each letter must appear in the document 10 times. If the margin of error is 10%, it's acceptable if a appears 9 times and b appears 11 times. There is only about a 3% error, which well within 10%. The report will display a distribution graph.

Runs Test



Frequency is only one piece of the big picture. Another piece similar to frequency is runs. It's possible for a document to pass a frequency test and also have a particular pattern. The string “aaabbbccc” have the expected frequency for each alphabet, but each character has the same number of runs appearing one after another. For each character, the program records the number of runs and the frequency. The string, “aabbbacca” has two one-run and one two-runs for the letter a. The frequency of n-runs for each letter of the alphabet must follow a normal distribution within a margin of error.

Sums Test



By far, the most mathematically rigorous test is the sums test. The order of appearance is the most important aspect of the sums test. The ith character of the test string is compared with the jth character of the test string. The letter a will be less than the letter c because its ASCII value is less than c. The test keeps a tally (let's call this sum) if the ith character is less than the jth character. The j variable is then increased by one until it reaches the size of the test string. After the jth variable reaches the end, the ith variable is increased by one. When the ith variable reaches the end, it will be time to do the necessary calculation in order to determine if the test string passes or fails. Suppose the user inputs the string "abcd". The comparisons are:

a to b
a to c
a to d
b to c
b to d
c to d

The sum for this particular example is six (a < b, a <c, a < d, b < c, and c < d). Let s be the size of the test string and c be the tallied sum. Notice the number of comparisons are in a triangular sum. In this case, only half is needed. In order to obtain a z-score to compare using a normal distribution curve, it's important to obtain the mean and standard distribution.

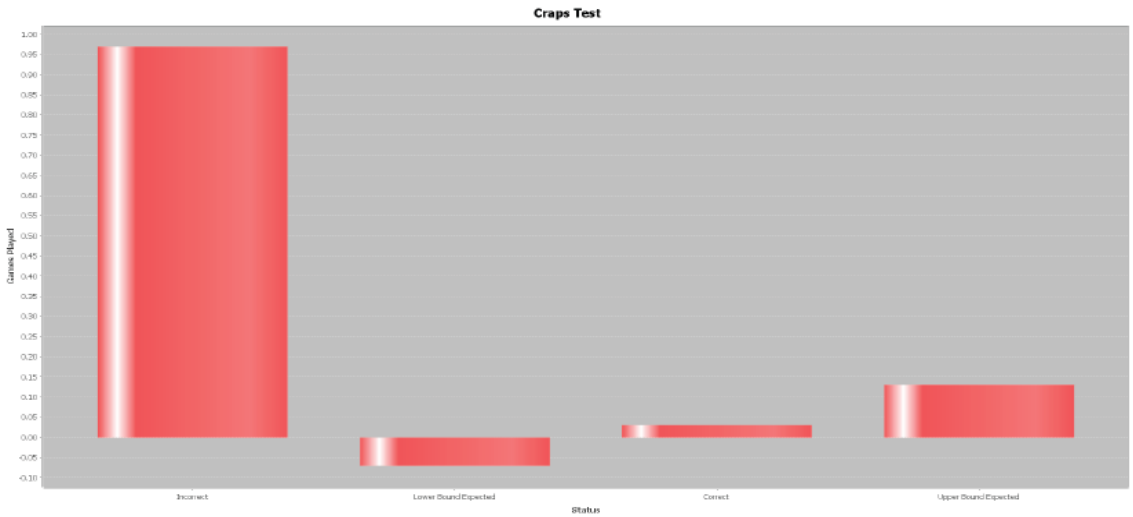
$$t=\frac{s(s-1)}{2}\frac{1}{2}=\frac{s(s-1)}{4}$$

$$\mu=c-t+\frac{1}{2}$$

$$\sigma=\sqrt{\frac{2s(s+5)(s-1)}{72}}$$

The chart in the report is a normal distribution chart. The blue line represents the margin of error and the red line represents actual value as computed by the normal distribution function using the z-score.

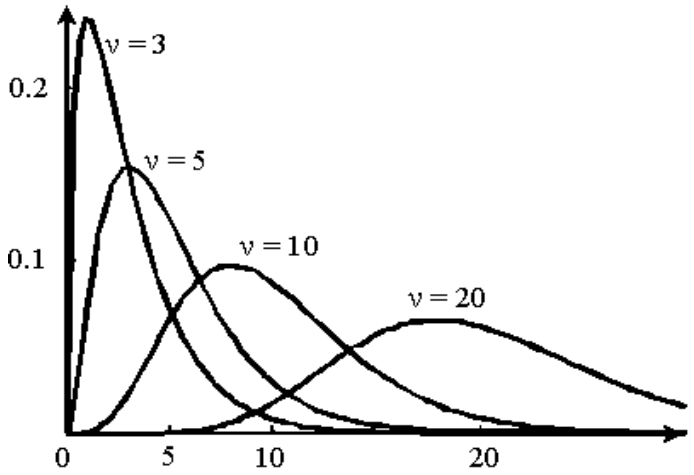
Craps Test



The craps test is the only unstable randomness test in

this program meaning that the test can switch between success and fail. However, it's mainly unstable when the runs and frequency distribution are more uniform. The program doesn't necessarily uses the classic game of craps, but instead it uses a somewhat similar notion. Any random generator can be used (in this case the standard java random generator is used). The program goes through each letter in the document in sequential order and the generator will produce a random number as a guess. The incorrect and correct guesses are tallied into percentages. If the percentage matches within margin of error 1/n (where n is the number of letters in the program's alphabet), it will pass, otherwise it will fail. The chart in the craps test contains four categories: correct, incorrect, lower bound expected, and upper bound expected. In order to pass, the correct value must be between the lower and upper bound.

Chi Test



One test is based on a uniform distribution and three tests are based on a normal distribution. However, the chi test is based on a chi distribution. The idea is to segment the frequency of each character in the program's alphabet into different number of boxes (a simple array) containing the frequency. In this case, the program segments the frequency into 10 boxes. The index of the box is the index of the character mod number of boxes. Take the string, “abcaaabcbcbac” and let's use 3 boxes.

A => box[0] = box[0] + frequency(A)
B => box[1] = box[1] + frequency(B)
C => box[2] = box[2] + frequency(C)
A => box[0] = box[0] + frequency(A)
A => box[1] = box[1] + frequency(A)
A => box[2] = box[2] + frequency(A)
so forth....

In order for the test to pass. It must be below a 5% on a chi curve. To figure the “score” a certain summation of each index of the box is used. Let z be the score used for the chi-curve, n be the number of boxes, s be the size of the test string, and B_k be the value of a box at k index. The number of boxes is also the degrees of freedom in the chi distribution.

$$z = \sum_{k=0}^{n} \left(\frac{B_k}{sn}\right)^2$$

Libraries Used

jfree charts – Generate charts for reports and distribution functions.
Apache pdf – Develop pdfs for reports.

Python – Particle Engine

Description and Motivation:
One of my long time hobbies is to study physics. I wanted to combine my love for both physics and programming, the result is a particle engine written in python using pygame as a graphical library. My main goal is to make the particle engine into a usable library for python developers who want to create physics based applications such as simulations and games. The particle engine class has a customizable data structure that allows the programmer to add, update, and destroy particles generators with ease. The user must launch a separate python program so he or she is able to input critical information such as velocities, accelerations, particle size, decay rates, and much more. The input of this program is a textfile in which the programmer use in his program. The information can be updated manually at anytime he or she wishes. Features includes automatic rotation, binding, and dynamic collision detection.

Technical Experience Gained:
1. Development of a graphic intensive application
2. Development of a library
3. Text processing
4. Algorithm optimizations

Details:

My ultimate goal is to create a highly customizable 2D particle engine. Users should be able to control critical aspects of a particle system such as color, duration, direction, range, speed, and much more. The particle engine should have routines in place that allow the initiation and termination of the particle generator at a given event defined in the user's code. Pygame library is used to implement graphical routines.

The major components of a particle system is:
1. Direction
2. Velocity
3. Acceleration
4. Position
5. Spread
6. Size
7. Color
8. Range
9. Rotation

1. *Velocity*

For simplicity as of now a single particle with be used as an example. Implementing the velocity of the ball is straight forward. The more pixels the ball moves in one frame, the faster it goes. Velocity is direction sensitive and applies to both the x and y axis. A negative x velocity means the ball is moving at towards the left and a positive x velocity means the ball is moving towards the right. However, a positive y velocity means the ball is moving angle upwards and a negative y velocity means the ball is moving downwards. In the program, this behavior is hidden from the user. Instead the user input an

angle from 0 to 360. These velocities are initial velocities, which will make more sense after discussing acceleration.

2. *Direction*

It's important for particles to be able to move in directions other than in straight lines towards the left, right, up, and down. The user must define one angle. The cosine value on a unit circle corresponds to the x value anywhere on the unit circle and the sine value corresponds to the y value. It's helpful to think of an angle as a percentage of rise and run. The cosine of an angle can be seen as the percentage of horizontal run (hence x value). Likewise, the sine of an angle can be seen as the percentage of vertical rise. To make sense of this, think of throwing a ball at a ninety degree angle. You're throwing that ball straight up in the air. Using degrees, input cos(90) and sin(90) into a calculator. You will receive 0 and 1 respectively. You can interpret this result as a 100% rise upwards (sine corresponds with y values) and a 0% run rightwards (cosine corresponds with x values).

3. *Acceleration*

It's unrealistic to have a particle moving at the same speed at any given time. Like velocity, acceleration has a x and y component. Usually in projectile motion, the acceleration for the y component is -9.8 m/s and 0 m/s for the x component. The reason is that earth's gravity is pushing objects downwards, but not at any x direction. In a graphic based applications, the number -9.8 m/s will most likely be scaled to realistically simulate earth-like gravity in a non earth like setting. The initial velocities are independent from the acceleration. Suppose I want to simulate a ball drop off the Eiffel Tower. There are no vertical or horizontal initial velocities involved. However, the forces of gravity (acceleration component) are acting on the ball causing it to go in motion despite having no initial velocities.

4. *Position*

Now everything is needed to track the x and y location of the ball. There are two separate classical physics formulas for final x and y distances.

$$x_f(t) = 0.5 a_x t^2 + \cos(\Theta) v_x t + x_i$$
$$y_f(t) = 0.5 a_y t^2 + \sin(\Theta) v_y t + y_i$$

So where do these equations come from?

Acceleration is the change of velocity (dv) over time (dt).

$$\frac{dv}{dt} = a$$

Velocity is the change of distance (dx) over time (dt). To get from the change of velocity over to the change of distance, integrate acceleration leaving the constant, v. (this is like adding + c after integrating)

$$\frac{dx}{dt} = \int \frac{dv}{dt} = \int a = a t + v$$

$$\frac{dx}{dt} = a\,t + v$$

The last step is getting to the actual distance itself, so integrate again. This leaves the constant x_i, which will be helpful in discussing initial distance.

$$x = \int \frac{dx}{dt} = \int a\,t + v = 0.5\,a\,t^2 + v\,t + x_i$$
$$x = 0.5\,a\,t^2 + v\,t + x_i$$

If we can look at x and y as separate entities having its own velocity and acceleration. Thus allowing us to split the equation into x and y parts.

$$x_f(t) = 0.5\,a_x\,t^2 + v_x\,t + x_i$$
$$y_f(t) = 0.5\,a_y\,t^2 + v_y\,t + y_i$$

Think of an angle as percentage in a general direction (which is defined as the sign part of the initial velocity). The goal is to go somewhere between the general direction. To do that, multiply the angle by the initial velocity and you will receive part of the general direction. The resulting velocities are:

$$\cos(\Theta)\,v_x\,t$$
$$\sin(\Theta)\,v_y\,t$$

Adding the initial distance is important to keep track of the exact coordinate of an object at any given time. If someone drop a ball off of a 500 m building and I want to find the y coordinate at three seconds, it makes sense to add the 500 m to offset the distance. Putting everything together we finally have:

$$x_f(t) = 0.5\,a_x\,t^2 + \cos(\Theta)\,v_x\,t + x_i$$
$$y_f(t) = 0.5\,a_y\,t^2 + \sin(\Theta)\,v_y\,t + y_i$$

Implementation:

Fortunately, the implementation isn't as involving as the concepts. Since pygame have built in timers, it's easy to keep track of time and retrieve the (x,y) position at any given the time. It's important to decide on interval of time as a constant. The interval of time is added to the overall time after each frame runs. If a user wants each particle generator to run in slow motion, he or she must set the time interval to a smaller amount.

Projectile Motion Algorithm:

INPUT: vel_x, vel_y, acc_x, acc_y, angle, x, y, time_interval

STEP 1: Add time interval to the overall time variable.

STEP 2: Input initial velocities, acceleration, initial positions, and angle into the x and y location formulas (as presented earlier).

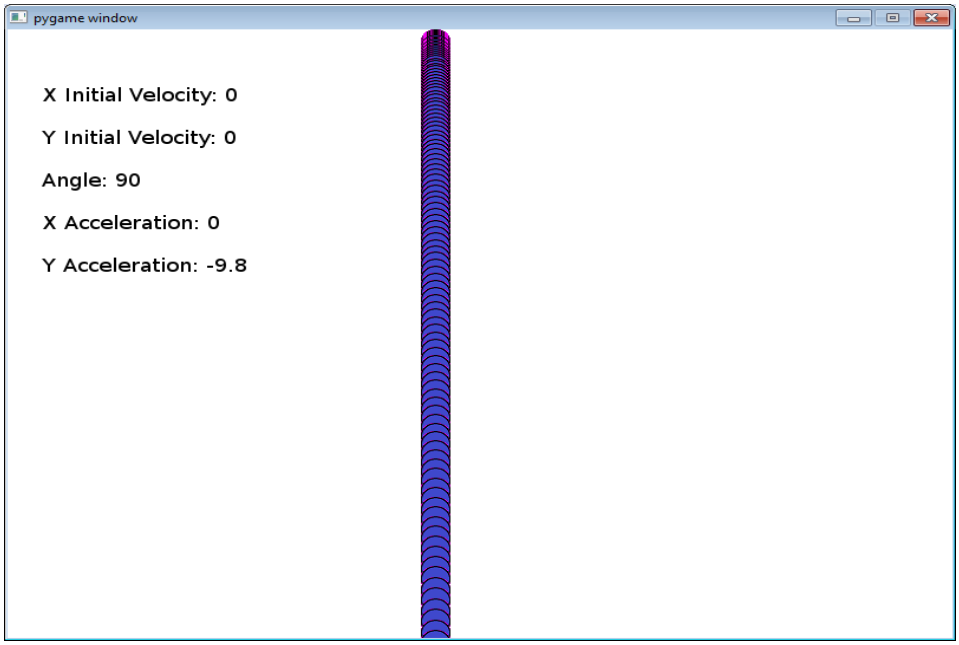
STEP 3: Update the location using the results from step 2 and draw the ball.

STEP 4: Repeat step 1 until user exits the program or implements a stopping conditional (ex: ball hits the ground).

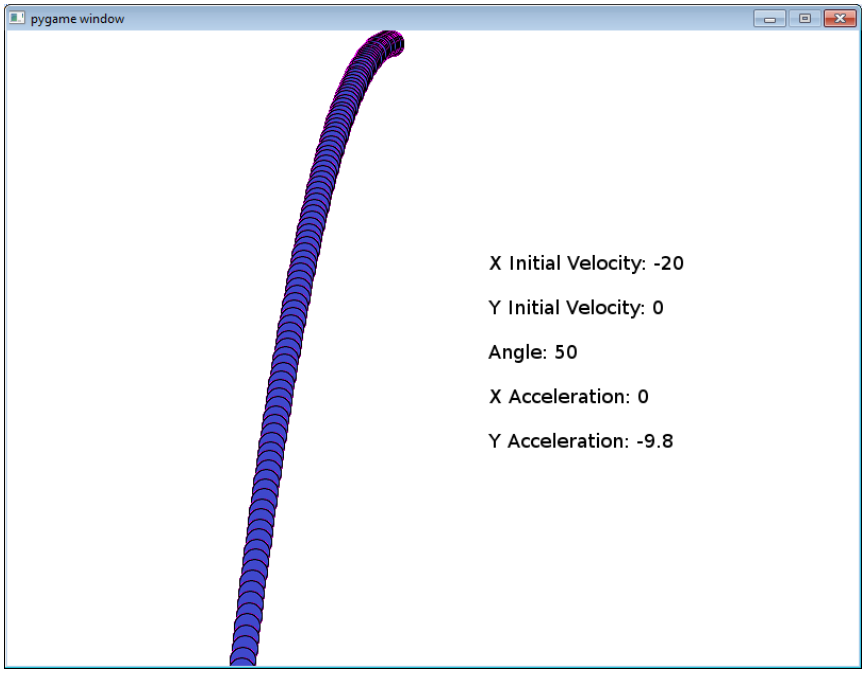
Note: the examples will have trails to visually display acceleration. The program provided on the website will have this disabled.

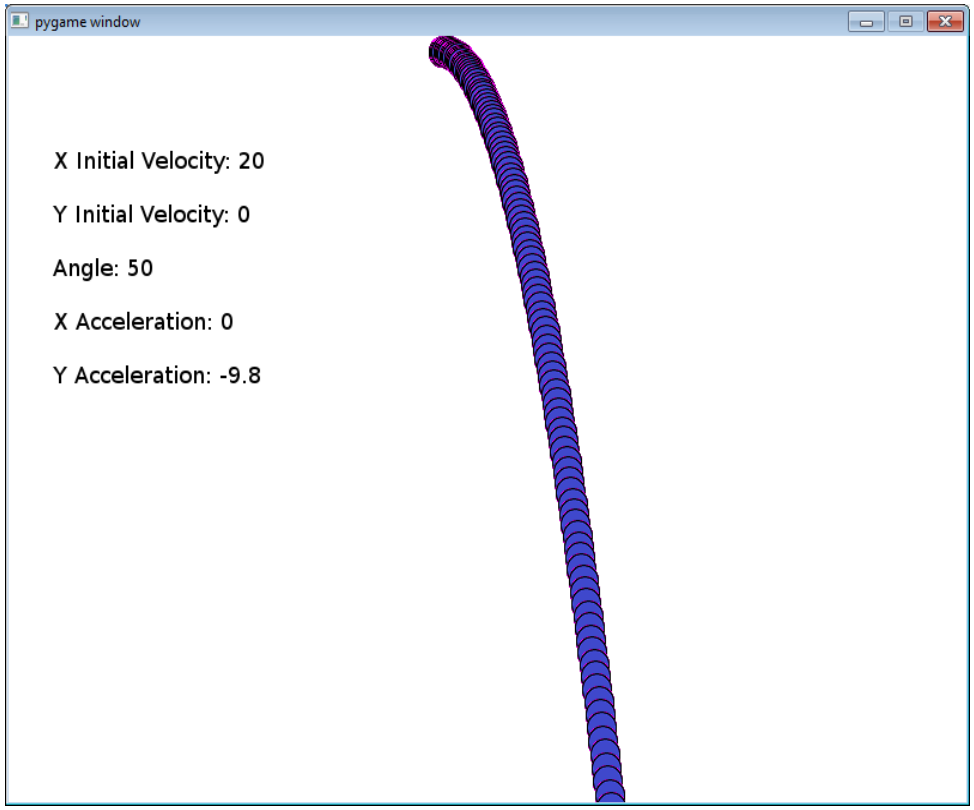
Examples:

Example 1: Free fall (i.e. dropping a ball off the Eiffel tower)



Example 2 and 3: Shows how the initial velocity controls the general direction through its signs.

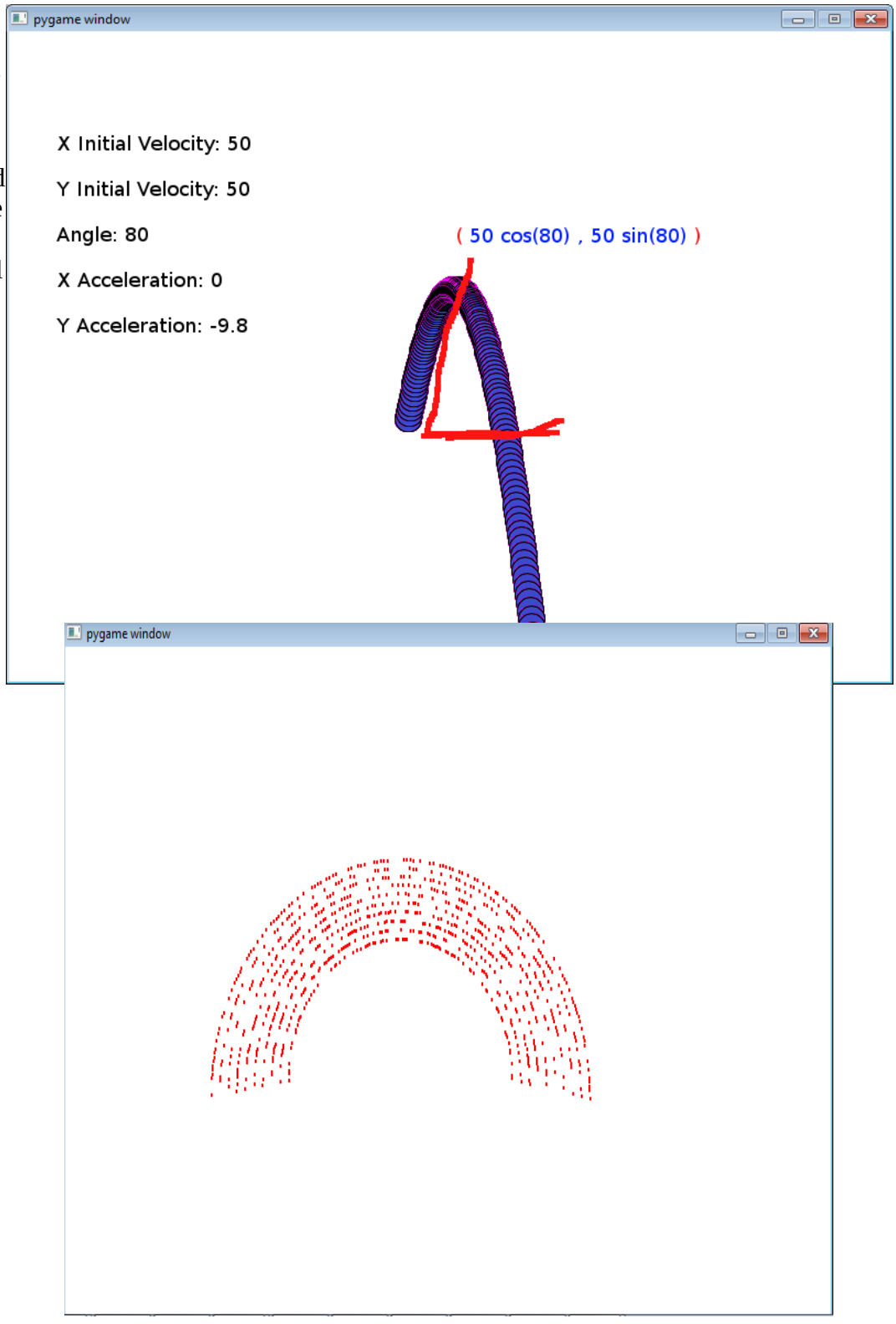

pygame window
X Initial Velocity: -20
Y Initial Velocity: 0
Angle: 50
X Acceleration: 0
Y Acceleration: -9.8


pygame window
X Initial Velocity: 20
Y Initial Velocity: 0
Angle: 50
X Acceleration: 0
Y Acceleration: -9.8

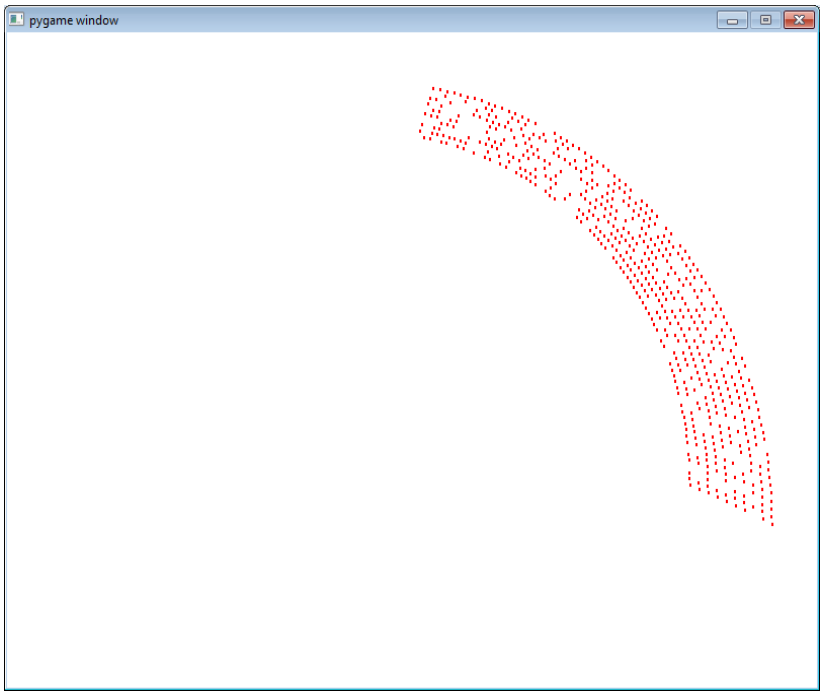
Example 4: More parabolic motion where the two initial velocities are equal. Also shows how an angle can control where the ball goes given a general direction (i.e. initial velocity).

5. *Spread*

As mentioned above, the user is responsible for inputing the angle of the projectile. The input will be a tuple of two values.



The screenshot above indicates the particle projects in a semi-circle fashion. Suppose that the user want the particles to flow in a semi-circle fashion on the bottom part. Users can achieve this by imputing the value [180,360]. The first index of the spread tuple doesn't necessarily has to be smaller than the second index. One case is that a user wants the particles to flow from the fourth quadrant to the first. An example of this would be [280,50].as shown below.



6. *Size*

The size parameter is straightforward. The program uses the transform pygame routine to transform each particle. The input is in the form of a tuple. The first index is the width and the second index is the height. The bigger the size, the slower the program will run. In most cases, if users want large particle sizes then they will usually want fewer particles for aesthetic reasons. Also, fewer particles balance the large size.

7. Color/Image

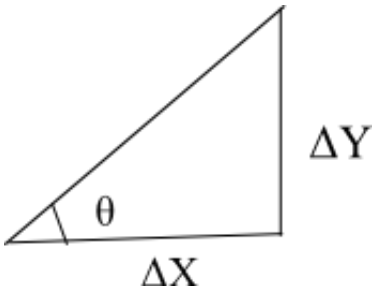
The color is a simple three size red, blue, and green tuple. If the user indicates a file name pointing to an image, then the color will be ignored and each particle will take the appearance of the image.

8. *Range*

It's important to establish a boundary to inhibit particles from going to unwanted locations and for optimization reasons. The range is a two size width and height tuple.

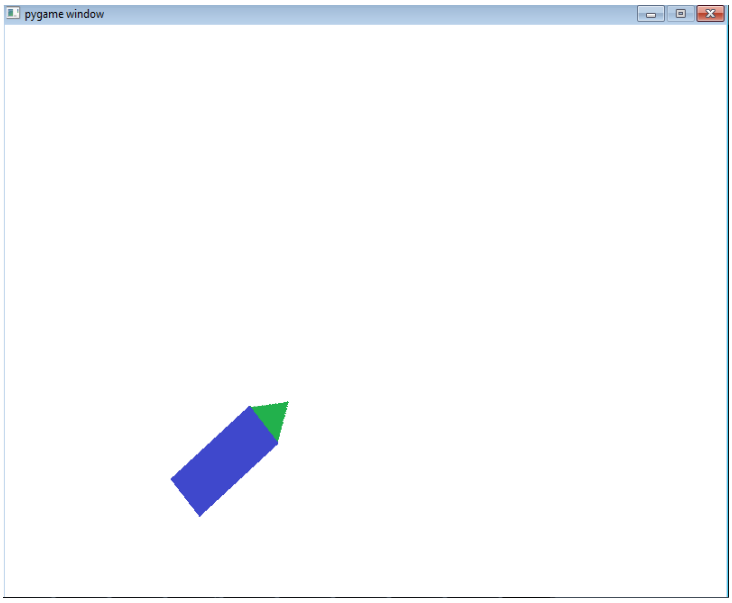
9. *Rotation*

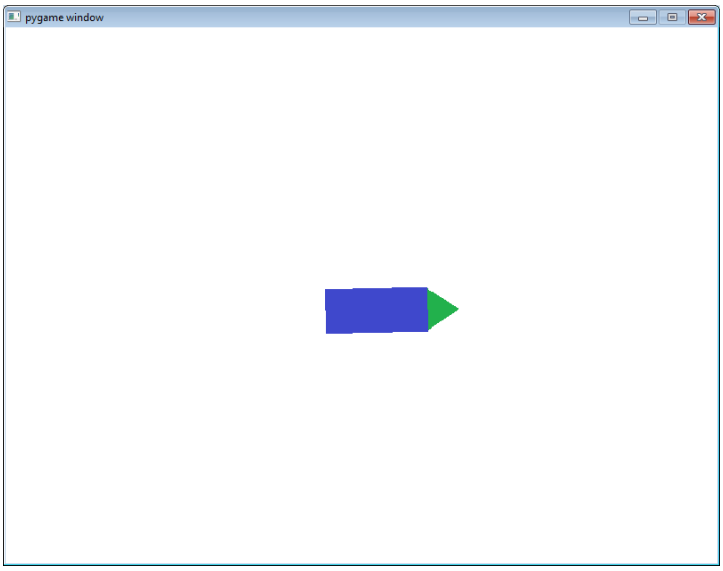
When the user uses an image for the particle and request rotation in the configuration file then dynamic rotation is used. First the user must set the image where it's at 0 degree angle. This can be achieved through a rotate feature in a simple image editing program. Two coordinates are recorded, the previous and the current coordinate. The goal is to use these coordinates to find the angle in which the image will be rotated. A trigonometric trick shown below is used to achieve this

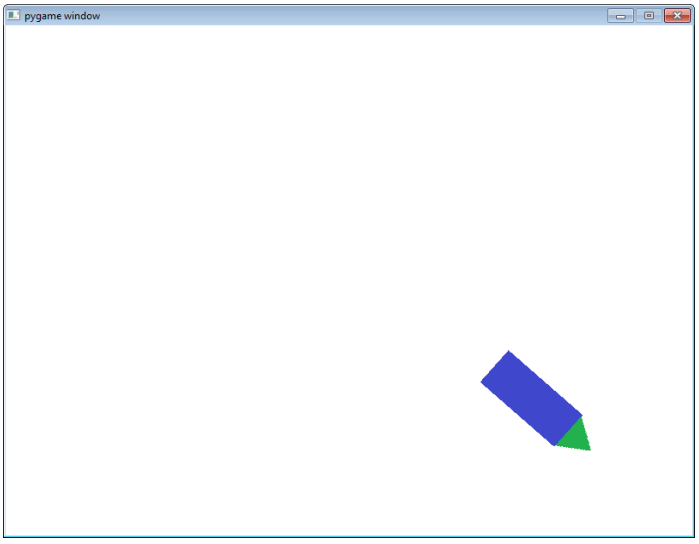


$$\theta = \tan\left(\frac{\Delta y}{\Delta x}\right) = \tan\left(\frac{y_p - y_c}{x_p - x_c}\right)$$

Result:







Data Structure

Since the users will mostly be programmers, a strong easy to implement data structure is essential.

Loading the script:

```
    part_sys.load_from_script("setup.txt",(350,350))
```

This will add a generator to the particle system. Each generator is contained in a list of generator inside of the particle system.

Activating the generator:

```
part_sys.activate_generator("rocket")
```

Updating each generator (or you can set individual generators in the parameters). The first two parameters are where the user want to draw the particles and a common variable used in pygame application that keeps track of the timing:

```
part_sys.update_generators(screen,ticker)
```

Python – Game for Software Engineering Course

Description and Motivation:
This game project is assigned to a group of four people. In addition to develop the code for a large project, we as a group, were required to create UML diagrams (Use Case and Class diagrams) and user documentation. The game is a 2D platform game where a user guides a character across multiple obstacles.

Technical Experience Gained:

1. Development of a graphical application
2. Multiple uses of polymorphism and inheritance
3. Text processing
4. Creation of UML diagrams: Use case and Class
5. Creation of user documentation
6. Developing in a team consisting of five people
7. Algorithm optimizations
8. Multiple uses of queues and dictionaries
9. Presenting technical and user details in a group of about 30 people for 20 minutes.

Reference:

Dr. Jiling Zhong
Email: jzhong@troy.edu
Phone: 334-670-3388
Title: Computer Science Professor

Terrence Lewis (Team Leader)
Email: NA
Phone: 334-268-8977

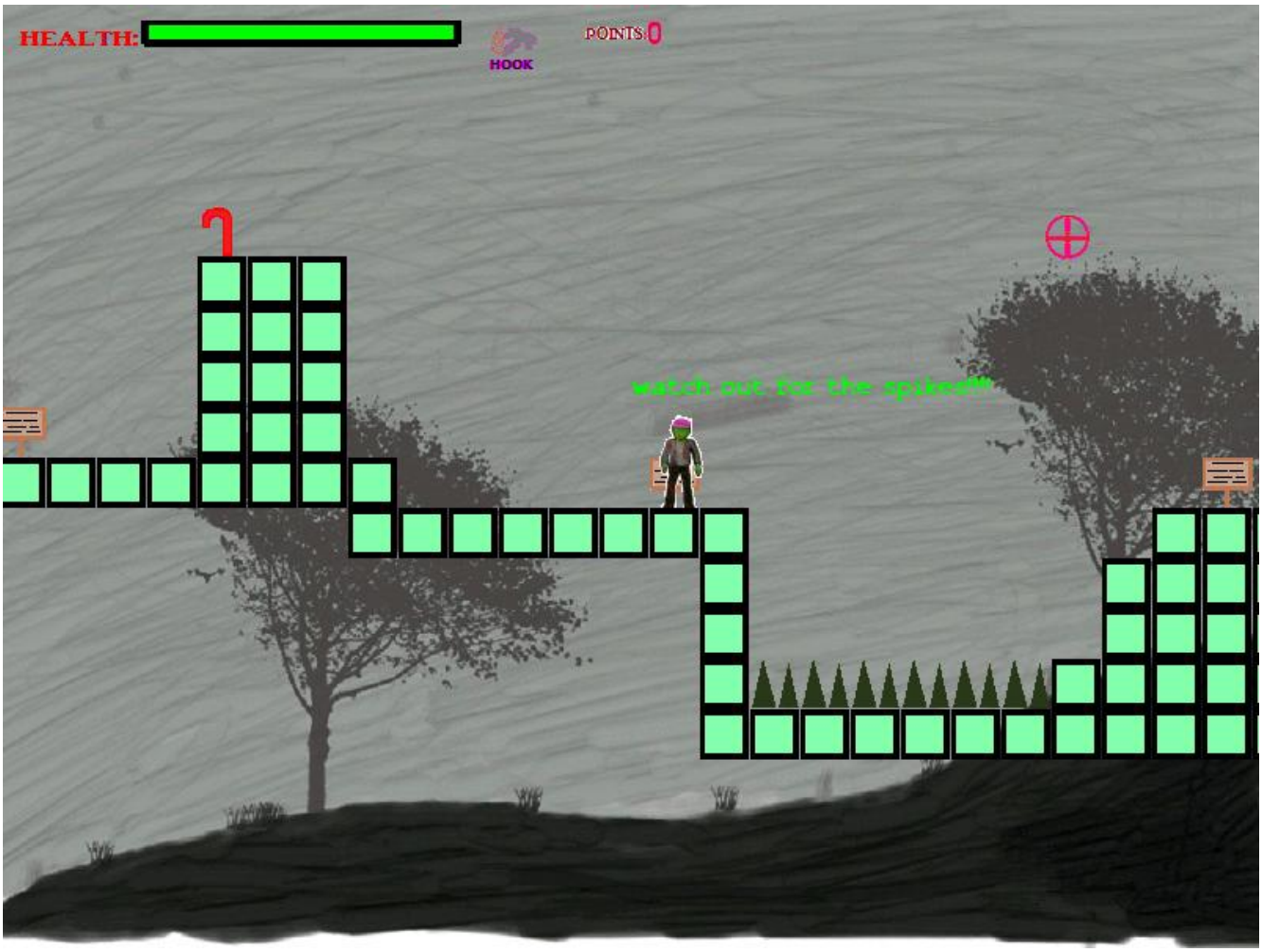
Details:

My Role

Since we were assigned four people, it was logical to divide the tasks for this particular project in four ways: documentation/presentation, programming, leader/design, and graphics. I took the programming role. I wrote a vast majority of the code and developed UML schematics of the program. This requires me to communicate with the three other team members so we as a group can get a idea of what we can and cannot do within a certain time frame.

Concept

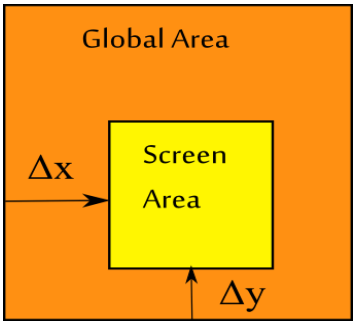
The group leader, Terrence Lewis, developed the concept of the game. It was a 2D platforming game where the user guides character through certain area filled with obstacles and progress to another slightly more difficult area.



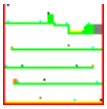
Mapping

A map is made up of many cells. Each cell is filled with small graphic usually referred to as a tile graphic. An object unlike a tile is a highly interactive function in the game (usually obstacles) that does something when a user triggers an event. A tile graphic can have two properties: solid and soft. A character or an object can rest upon a solid tile, but overlaps through a soft tile. The map is a 2D list made up of cells with a fixed width and height (usually 32x32 or 16x16).

A tilesheet is a pallet of tile graphics like the image above. The value of the list is the ID of the tilegraphic. Suppose the 2nd graphic of a tilesheet is a tree. To draw a tree in the 3rd row and 6th column, it would look something like this: map[2][5] = 2. A problem arises whenever the program attempts to draw tile on a map larger than the screen.



The whole map is known as the global area. The global area is usually much larger than the screen size. Inside the global area exist a screen area where the user is currently viewing. Suppose the user has a 1024x768 screen and the map has the size 5000x5000. The screen is located at (300,500) inside of the global area. When object is drawn onto the screen, it uses the screen coordinates. However, since the entire map is larger than the screen area, it's not logical to use screen coordinates to do calculations so it's better to use coordinates relative to the map and not the screen. Once the character moves past a certain point, delta x and delta y increases and new images are drawn onto the screen and old images are disposed from the screen. In other words, the map scrolls as the character walks through a certain point (in this program, that point is the center of the screen). The delta x and delta y plays a pivital role for mapping objects. Suppose an object is drawn on the screen at (100,700) (screen being (300,500)). The global coordinates for that object is (100+300, 500+700) or (400, 1200).



Also due to time constraints, a fully functional map editor would not be possible. Instead the designer can use a simple paint program to plot objects and tiles without even viewing the source code. The image above is an example of a map scheme. Each color represents a certain tile or object. The white pixels are empty spaces. As the program loads its resources, it scans the map scheme and plots the tile image at that location (except on a larger scale) based on the pixel color. A common problem is locating the column and row of a cell that contains a given point. Instead of cumbersomely scanning each cell until the point is found, using floors to find the point is a much more efficient method. Finding the row and column of a particular point requires just one formula. Let w be the width of the map in tiles and h be the height of the map in tiles.

$$R=\lfloor \frac{x+\Delta x}{w} \rfloor$$

$$C=\lfloor \frac{y+\Delta y}{h} \rfloor$$

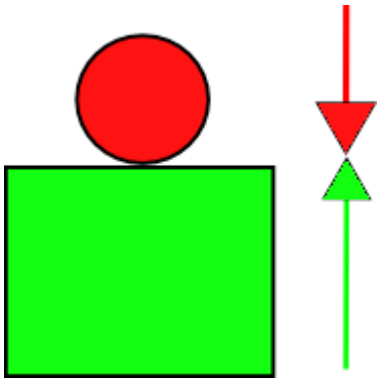
Loading and Saving

Before a user can operate the program, all resources must be loaded appropriately and efficiently. To avoid needless hassle, the designer (draws the map schematics and plot obstacles) should never have to add elements such as the coordinate of a certain obstacle directly into the code. Due to time constraints it was unreasonable to provide a seperate GUI tool to make this process easier as planned. However,

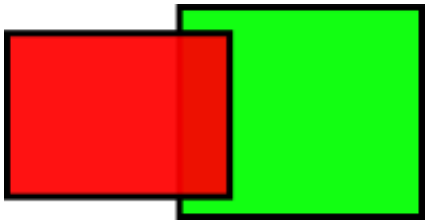
editing textfile sufficed. The code for loading attempts to hierarchically search through map files. Suppose the designer wants to add a fourth map: he would create a folder named map4, run the program, and the program creates empty textfile where he would then use it to design the map. The program would also append a new instant of a map to the array of map files when it encountered the map4 folder with the necessary information. The user can save it's progress during operating the game. This is where the “pickle” python module comes in handy. The game class which contains everything needed to operate the game is saved onto one file. When loading from the save state, the program uses the pickle module to unload everything at once.

Collision Detection

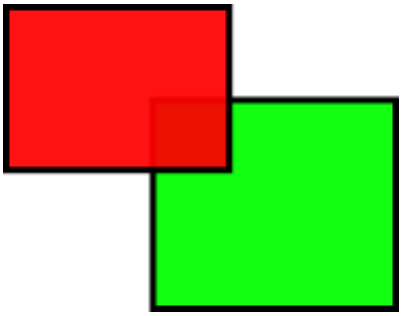
When two objects (or tiles) overlap, a collision occurs. A dynamically bounding box collision detection is implemented in this program. Each object are formed into rectangles and when the rectangle overlaps a collision event is triggered. For soft tiles, such collision is ignored. However for hard tiles, the object must react appropriately when a collision is triggered.



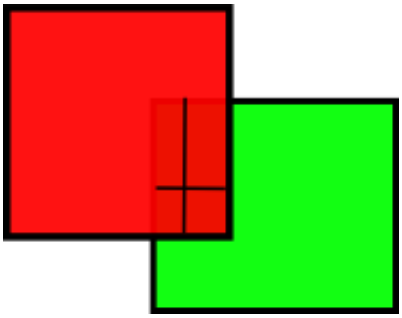
Since this application is a platform game, it's important to have gravity pushing downwards on objects. So objects are always trying to move down, but are usually held up by a platform (in the form of a tile). In this case, collision from the top of the platform to the bottom of the object are always occurring. A force acting on an object will have an opposite and equal reaction applies true to collisions. The collision of the object going downwards due to gravity and onto a platform causes the object to move upwards. Since this process is always occurring, the object will not bounce up and down, but instead stand still. By far the biggest challenge of dynamically detecting collision is identifying, which side is approached. It's illogical for an object approaching to the left side of the platform be projected upwards.



Take the above collision for an example. It's very clear that the rectangle on the left collided with the rectangle on right from the left side of the right rectangle. The reaction would be that the rectangle on the left move to the left so it's no longer colliding with the rectangle on the right.



Some times it's unclear where the object is approaching at. In the example above, the left rectangle maybe approaching the right rectangle from the top or from the right.



The solution to these problems is to examine the horizontal and vertical offset. To determine what direction to project at, the program compares the two offsets. If the horizontal offset is less than the vertical offset (as in the case of the example above) then the object will be projected from the left or right at that offset amount. Likewise, if the vertical offset is less than the horizontal offset, the object will project either up or down by that offset amount. The next step in this particular example is to determine if the object project left or right. The x location of the left rectangle is more to the left relative to the center of the right rectangle. Therefore, the left rectangle will project to the left. The third case is that the vertical and horizontal line are equal. In such case, the box moves in both direction in the offset amount.

Animation

An animation graph strip looks like the one above. The graph is split into 10 32x64 parts representing an individual frame. Each frame is inserted into a list. Whenever a user triggers an event (in this case, press the left arrow key), an index of the image displayed increases at a rate, “n mod 9”.

Intractable Obstacle

An intractable obstacle is a obstacle in the game that responds to user's action. One obstacle for an example, follows your character horizontally and another obstacle throws projectiles when your character is in a certain range. Each obstacle is controlled through a text file. Below is an example of a

configuration of these obstacles in one map.

```
E0_TYPE=lrgunner
E0_IMAGE=resources/esrt/lrgunner.bmp
E0_NUM=8
E0_I0_RANGE=0
E1_TYPE=moving_gunner
E1_IMAGE=resources/esrt/moving_gunner.bmp
E1_NUM=5
E1_I0_RANGE=50
E1_I1_RANGE=50
E1_I2_RANGE=50
E1_I3_RANGE=50
E1_I4_RANGE=50
E1_I5_RANGE=50
E2_TYPE=soldier
E2_NUM=4
```

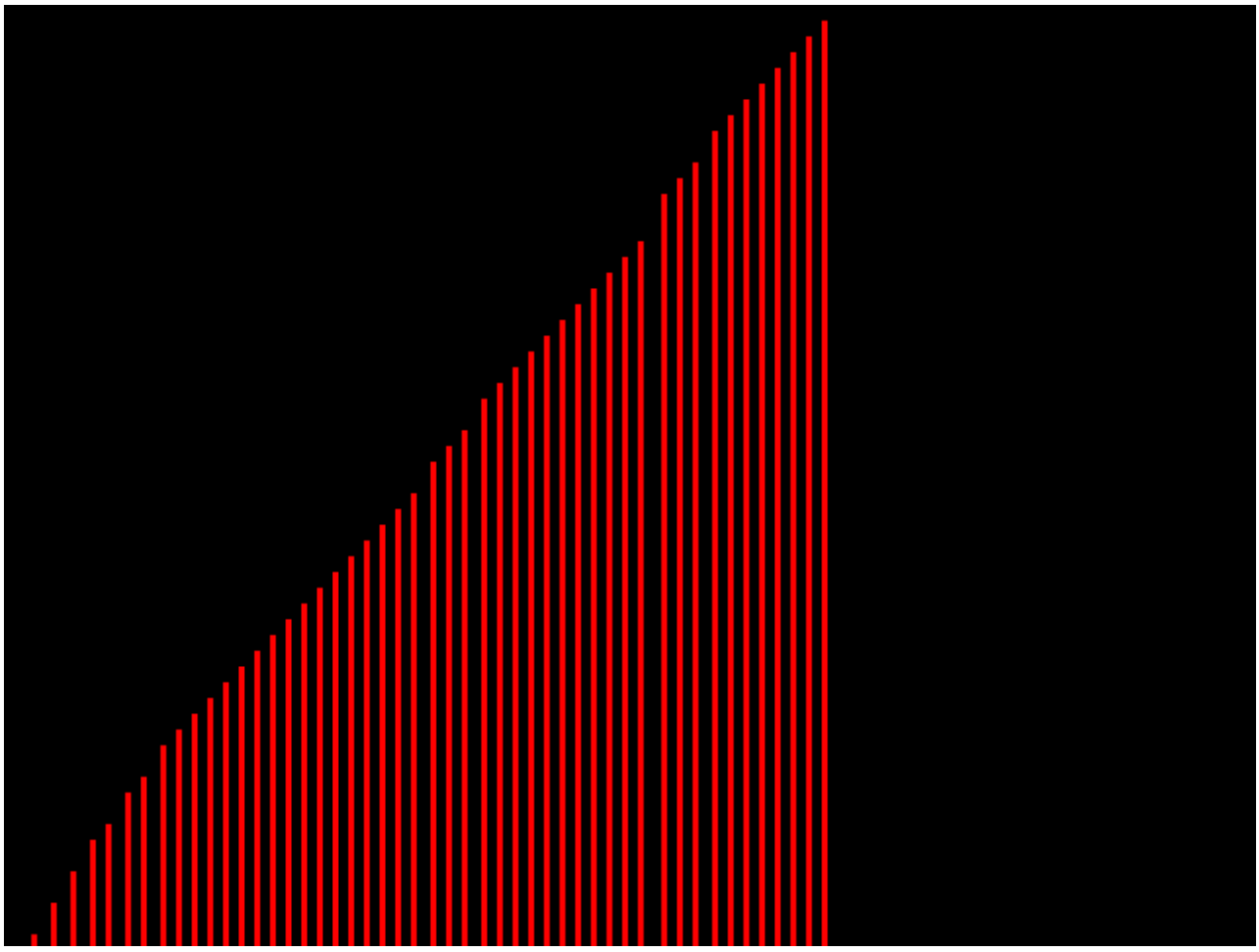
Minor Projects

C++ - Sorting Animation

Description and Motivation:
I became interested in various sorting algorithms and how they work in a more visual level. I started out implementing each sorting algorithm, I have learned in Data Structures and Algorithms class. Then I modified the program so I can visually show each step of the process using bars. A very popular and similar demonstration can be found for Java (http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html).

Technical Experience Gained:
1. O(n) and O(n log n) sorting algorithms
2. Animation and graphics programming

Details:
Source code provided on my website that includes implementation of each algorithm.



Keys:
ESC – Exit

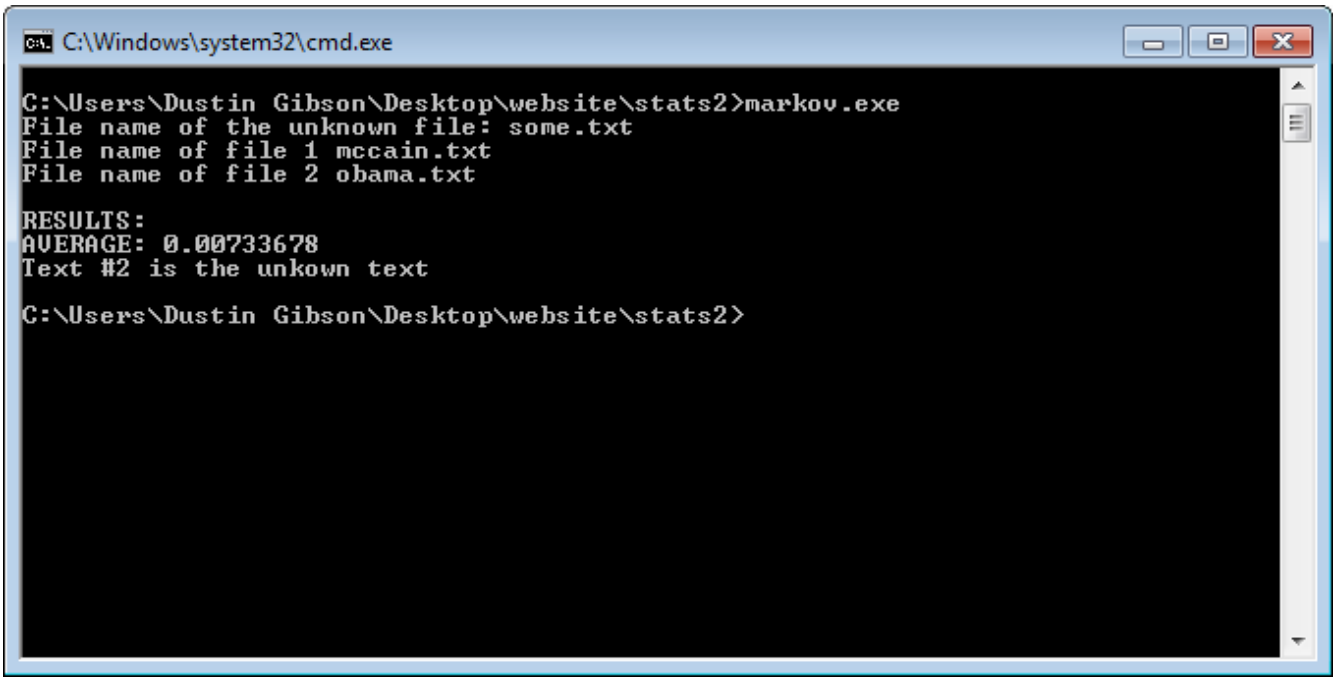
R - Reset
B – Bubble Sort
Q – Quicksort
I – Insertion Sort
M – Max Sort
H – Heap Sort
S - Mergesort

C++ - Text Predictor

Description and Motivation:
One of my favorite areas in mathematics is statistics and probability. I wanted to apply my knowledge that I have learned from probability theory and from C++ programming class. This project is similar to this programming assignment (http://www.cs.princeton.edu/courses/archive/spr05/cos126/assignments/markov.html). The user enters three files: first known text, second known text, and an unknown text. The goal is to predict the author of the unknown text using known samples.

Technical Experience Gained:
1. Markov chains
2. Text processing

Details:



First each text (including unknown text) is stripped of tabs, extra space, and page breaks. The program records frequency and runs of two character sequences. For an example: ababdacd is gathered as ab, ba,ab,bd, da, ac, and cd where ab is recorded as having a frequency of two. The same is repeated for three character sequences. A logarithmic probability value is associated for each three sequences and its two "sub-sequences" (abc → ab, bc). Let p be a set of probability values associated with sequence index, i for the first known text. Let q serve the same function as p, but for the second known text. Let n be the set of matching two sub-sequences from the known text to the unknown text. Let m be the same for three sequences. m and n applies only towards the first known text. r and s holds the same purpose as m and n except it applies towards the second known text.

$$p_i = \log\left(\frac{n+1}{m+3}\right) \quad q_j = \log\left(\frac{r+1}{s+3}\right)$$

Now, the summation of these probabilities are taken account for the overall logarithmic probability that will ultimately determine which text is most likely attributed to the unknown text.

$$L = \sum_{i=0}^{|p|} p_i - \sum_{j=0}^{|q|} q_j$$

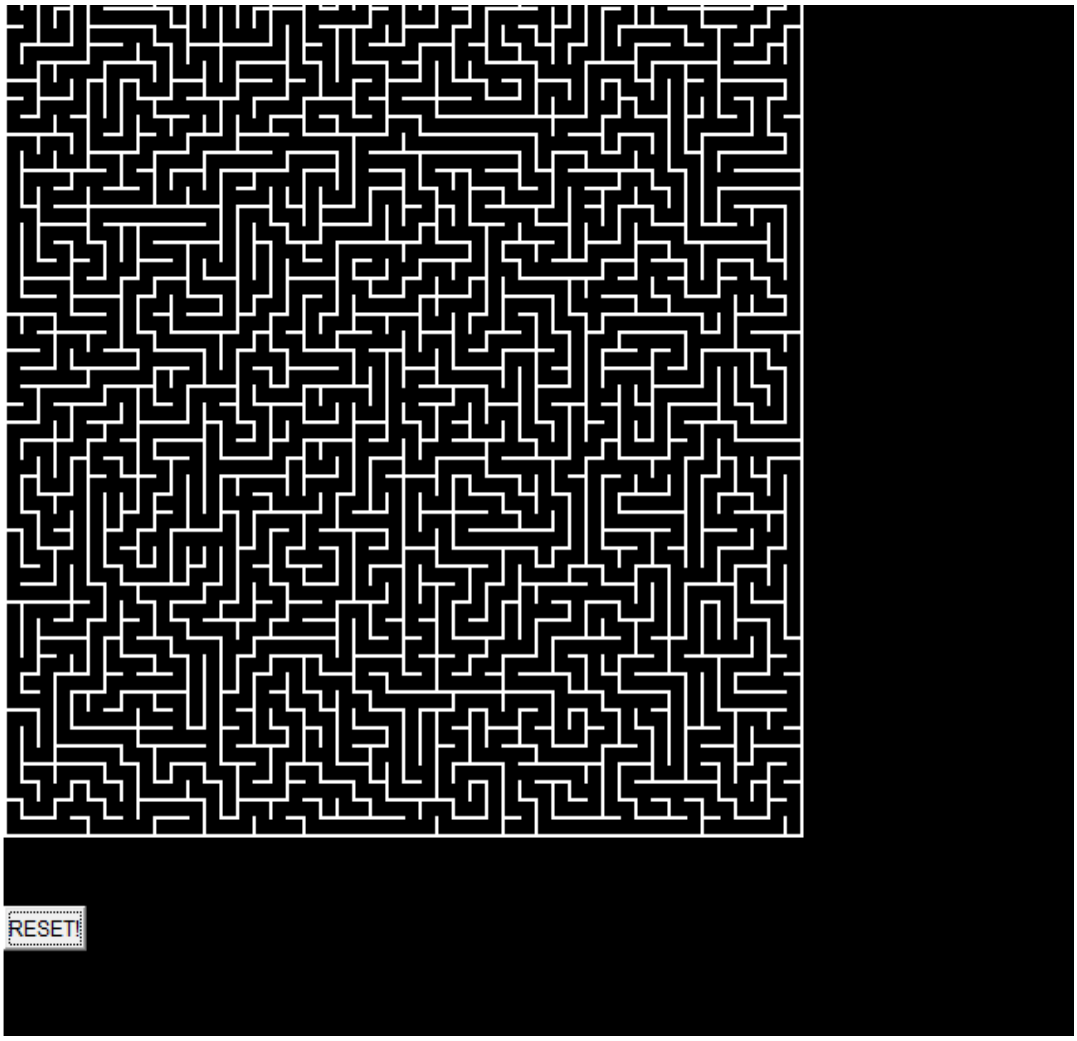
If L < 0 then the author first text is most likely the author of the unknown text
If L > 0 then the author of the second text is most likely the author of the unknown text
If L = 0 then both text are identical

Java – Maze

Description and Motivation:
I was interested in graph theory applications and used a recursive solution of the depth first search algorithm to implement randomly generated maze.

Technical Experience Gained:
1. Traversal graph algorithm (DFS)
2. Use of graphics in a Java application
3. Creation of Java web applet

Details:



The program begins with a grid consisting of n x n cells. Each cell has a north, south, east, and west wall. The goal is to erase these walls to create paths that is consistent with a maze. In order to accomplish this goal, each cell must have at least one box removed.

INPUT: n size (width and height will be identical). Let x and y denote cell row and column respectively. x=y=0

STEP 1: Create a grid consisting n by n cells. Each cell has a south, east, west, and north wall. Two or more cells may share a wall. Take two cells stacked vertically as an example: the top cell south wall is the same as the bottom cell north wall. If the top cell's north wall is removed then the bottom cell's south wall will be flagged as removed. In addition, the border walls (the upmost, bottommost, leftmost, and rightmost walls of the maze) are flagged as removed so that the program cannot remove walls that are not suppose to be removed.

STEP 2: Put the walls flagged as “not removed” for cells located at x and y into a list. If the list size is 1, then removing a wall will make a “hole” in the maze. If this is the case flag the cell as “finished” go to step 5, otherwise go to step 3.

STEP 3: Randomly select an item from the wall list and remove the selected wall. Flag the appropriate walls of each cell as “removed”. If south is selected increase y by 1, if north is selected decrease y by 1, if east is selected increase x by 1, and if west is selected decrease x by 1.

STEP 4: For each cell, check if all walls cannot be removed. If this is true for all cells. End the program.

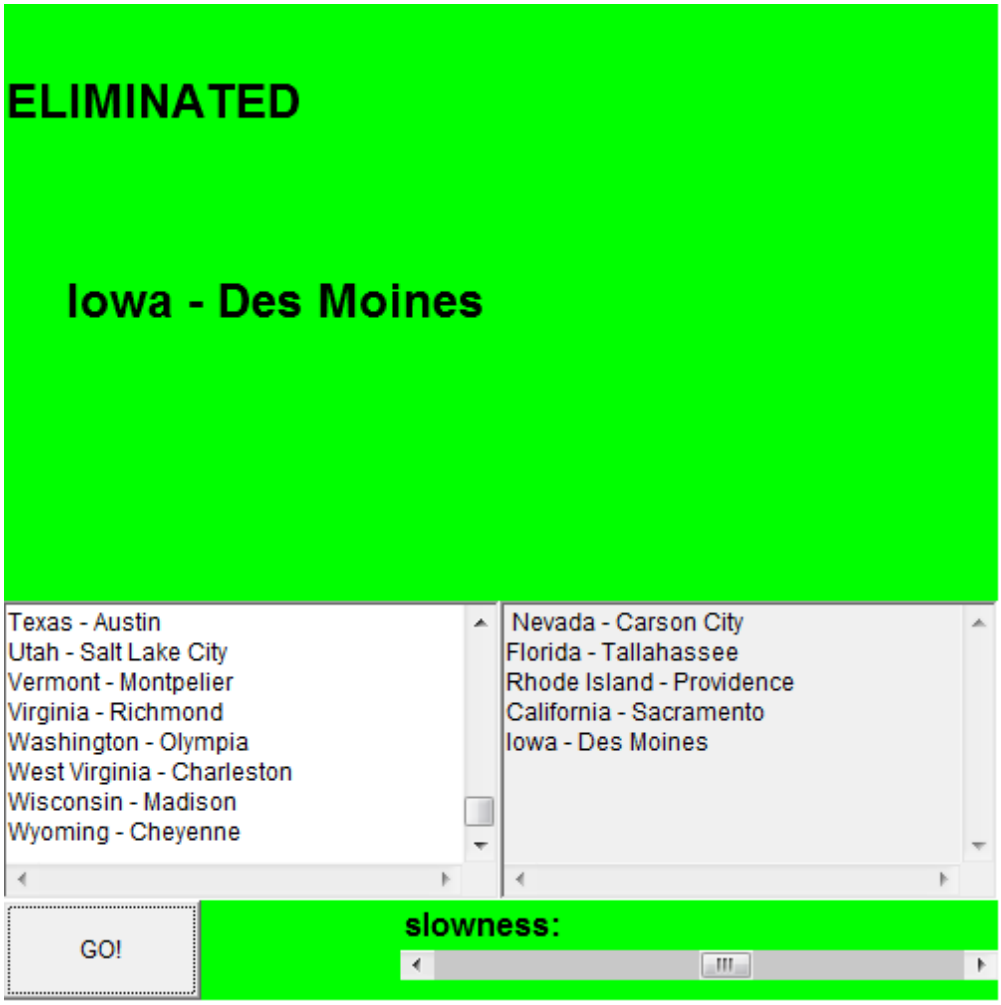
STEP 5: This step handles deadends of the maze. The program must backtrack into a cell with two or more walls. When the cell is reached go to step 2.

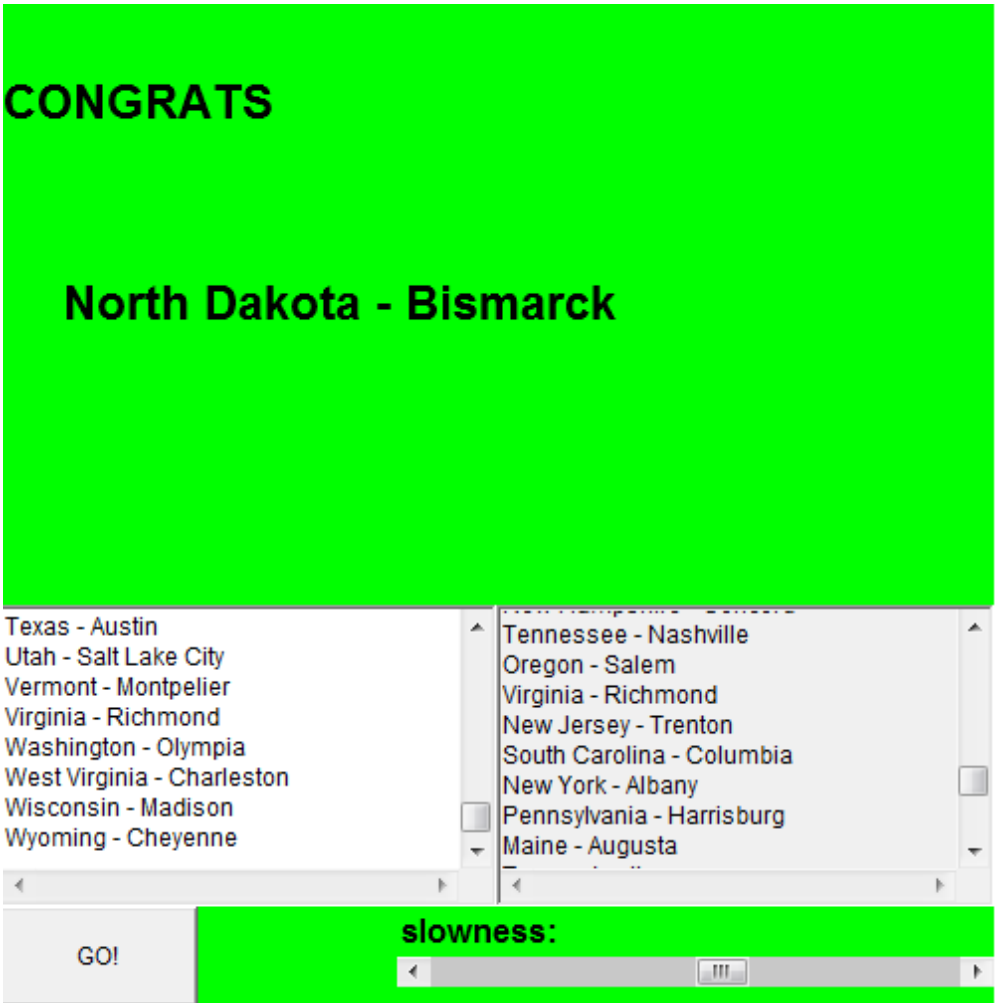
Java - Decider

Description and Motivation:
On an occasion I wanted the selection of the next movie I want to watch or book I want to read to be random. I created this program specifically for this task.

Technical Experience Gained:
1. Java web applet
2. Java graphics and GUI
3. Java threads

Details:





User enters a list of items on the left enabled textarea and presses go. Each item of the list is deleted one by one at a speed that can be controlled using the "slowness bar" located at the bottom right of the program. The eliminated items are recorded in the right disabled textarea in chronological order.

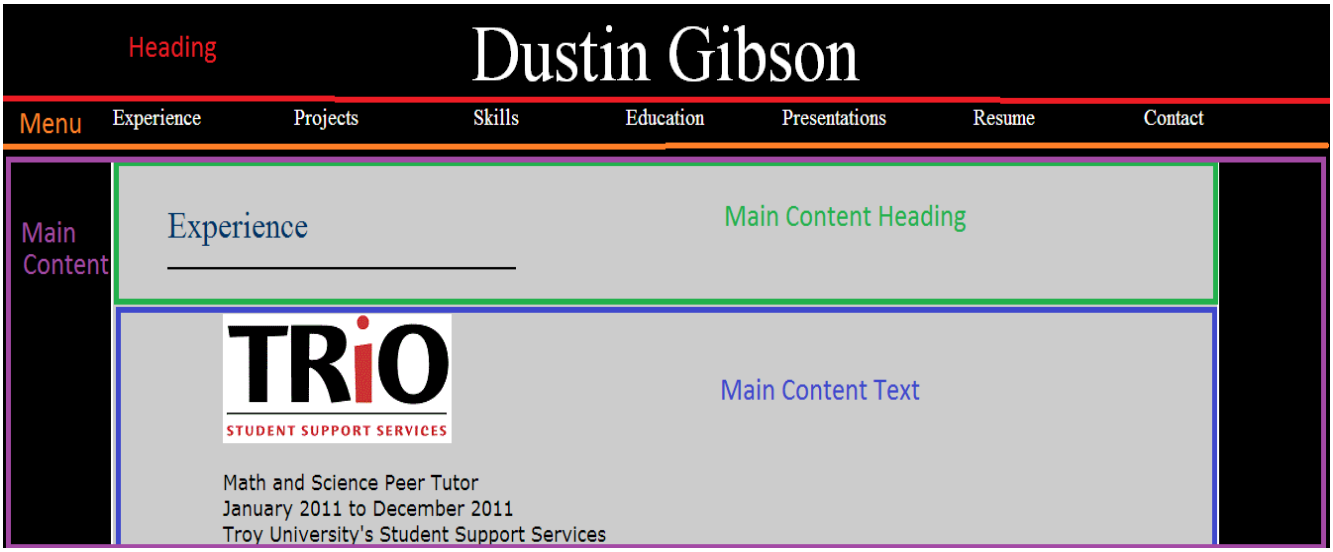
HTML/JS/CSS – Personal Website

Description and Motivation:
I wanted to create a site that extends my resume. Website can be viewed here:
http://bit.ly/dustingibson

Technical Experience Gained:
1. Website design and style with CSS
2. Menu building with JS and CSS

Details:

Divisions



Heading – Large white font consisting of the title of the webpage

Menu – A table consisting of a single row of links. When a user hovers over the items, the font color changes.

Main Content – Margins offset by 8% from both sides. Surrounded in a grey box with a solid white border.

Main Content Heading – Margins offset by 5% from the left relative to the main content division (13% total offset from the left and 8% offset from the right). Text follow by a heading line.

Main Content Text – Margins offset by 10% from both sides relative to the main content division (18% total offset from left and right)

Menu

The menu consist of a fairly trivial javascript code that overwrites the font color HTML when the user hovers in or out of a row in the menu table.

```
function SelectText(itemID)
{
	var prev = document.getElementById('item'+itemID).innerHTML;
	var over_write = "<span style='color:#ff0000'>" + prev + "</span>";
	document.getElementById('item'+itemID).innerHTML = over_write;
}

function DeselectText(itemID)
{
	var prev = document.getElementById('item'+itemID).innerHTML;
	var over_write = "<span style='color:#00ff00'>" + prev + "</span>";
	document.getElementById('item'+itemID).innerHTML = over_write;
}

function gotoPage(location)
{
	window.location.href = location;
}
```

Relevant Education

Troy University
GPA: 3.7/4.0 (Magna Cum Laude)
Major: Mathematics
Minor: Computer Science

Database Administration
Grade: A

1. Developed ER and enchanted ER diagrams to describe relationships between tables.
2. Determined restrictions on table entries including primary and foreign keys.
3. Wrote complex SQL queries to extract tables of data in a given database.
4. Used MySQL software to manage multiple databases and create a variety of queries
5. Converted relational algebra equations into SQL commands, vice versa.
6. Converted query descriptions into relational algebra equations.
7. Used several techniques to avoid redundant relationships among tables.
8. Implemented a database program in C#

Algorithmic Graph Theory
Grade: A

1. Graph data structures.
2. Breadth and depth first search.
3. Min/Max flows.
4. Greedy graph coloring algorithms.
5. Minimal spanning trees.
6. Planar graph detection algorithms.
7. Combinatorial analysis of graph coloring using chromatic spectrum.
8. Euler and Hamilton paths
9. Hypergraphs: hypertrees and chordal hypergraphs
10. Greedy hypergraph colorings.
11. Implemented a large program demonstrating graph algorithms

Data Structures and Algorithms
Grade:A

1. Lists: Stacks and Queues (including priority queues).
2. Used several mathematical techniques to analyze space and speed properties of algorithms.
3. Trees: binary, self balancing, B-Trees, 1-2-3 trees.
4. Sorting: merge sort, quicksort, bubble sort, insertion sort, and radix sort.
5. Graphs: traversal algorithms, data structures, and shortest path.
6. P vs NP algorithms.

Software Engineering
Grade: A

1. Presented and completed a large scaled programming project.
2. Software development models: waterfall, extreme programming, etc.
3. Design Patterns
4. Software Testing: unit testing, coverage, etc

Programming Courses
Grade: A (Programming Mathematics) and A (Computer Science II)

1. Used C++ to develop small scaled program mathematical applications.
2. Concepts in Object Oriented Programming: Polymorphism, inheritance, and composition.
3. Advance concepts in programming: templates, generic classes, and debugging tools.
4. Developed many small scaled programs to solve problems in mathematics.
5. Memory management and pointers.
6. Recursive vs iterative algorithms
7. File serialization

Programming Languages
Grade: B

1. Context-free grammars
2. Parsing using derivation trees.
3. Survey of programming languages: Ada, Scheme, Ruby, etc.
4. Inner workings of object oriented languages: ex: Static vs Dynamic binding.
5. Compilers and the pushdown automata.

Formal Languages and Automata Theory
Grade: A

1. Regular grammars and regular language.
2. Finite State Machines
3. Context-free grammars and pushdown automata
4. Using pumping lemma to prove a language is not context-free.
5. Turing Machines and the haulting problem.
6. Variation of turing machines.
7. Using turing machines to analyze P vs NP problems.
8. Chomsky hierarchy

Discrete Mathematics
Grade: A

1. Basic counting techniques.
2. Solving permutation and combination counting problems
3. Pidgin-hole principal
4. Develop programs that generate permutations and combinations.
5. Inclusion and exclusion principal and its applications.

Other Qualifications

Relevant Professional Experience
January 2011 – December 2011
Math/Science Peer Tutor
Troy University Student Support Services
Phone: 334-670-5985

Work Ethic:
I maintained a perfect record in punctuality. I have also went above and beyond my responsibilities by volunteering to meet students outside of the classroom for their tutoring needs. It's my personal belief to respect and be generous to everyone and complete tasks assigned by my superiors in a timely manner.

Communication Skills:
I clocked over 100 one on one tutoring hours as a math and science tutor in Troy University. It's vital for me to use verbal and non-verbal communication skills to help students succeed in their courses. I have also gave several presentations in a group of 3-20 students and sometimes staff members. Over time, my communication skills improved dramatically to the point where communicating complex ideas in a group and on a personal level became intuitive. I have also completed a course in speech and communication with an A average.

Other Technical Experience:
Occasionally students and staff members have difficulty repairing or troubleshooting problems they are experiencing with their computer and/or computer equipment. As a math and science tutor, one of my responsibilities is to help the math specialist (also my supervisor) maintain and install large software packages onto about twenty workstations inside of the Student Services labs. I have successfully installed large and maintained large software such as Deep Freeze, Windows 7 Operating System, and Microsoft Office Suit. Students personally called me for personal help with their computers. I have troubleshooted and fixed problems ranging from Hard Disk failure to malware problems I have also fixed and installed several large networked printers, copiers, and fax machines..

Clerical Experience:
I have also spent many hours as a tutor assisting the staff with a wide variety of clerical duties. The assignment includes: copying papers, filing student documents, recording data onto student documents, answering phones, faxing documents, guiding visitors to areas inside of the facility, and assisting my supervisor with monthly reports, and using excel to gather statistical data regarding to student usage of resources.