student_showcase

# from n00bs to ninjas in < 18 months

Bruce Fuda
@Bruce1979
The guy at the front...

Bruce Fuda
Director of Technologies

experience = [
    “teacher”: “12 years; government high schools and colleges, ACT”,
    “developer”: “web applications - school support; python/flask, HTML/CSS/JS”
    “curriculum”: “adviser and writer for Australian Curriculum: Technologies”]
the_teaching_program

# what students learn before embarking on their projects
A two year program of study across a choice of subjects

**Information Technology @ Gungahlin College**

**Year 11, Semester 1**

- **Computer Science**
  - boolean logic
  - state machines
  - nature of data

- **Programming**
  - Python
  - language fundamentals
  - functions

- **Web Design**
  - HTML / CSS / JS
  - design principles

**Year 11, Semester 2**

- **Computer Science**
  - algorithms
  - data structures
  - recursion

- **Programming**
  - object-oriented
  - event-driven
  - games programming

- **Web Design**
  - introductory Python
  - jQuery
  - databases

**Year 12, Semester 1**

- **Computer Science**
  - algorithmic efficiency / complexity
  - modelling data
  - combinatorics

- **Programming**
  - meaningful inheritance
  - “advanced” python
  - useful python modules

- **Web Design**
  - MVC pattern
  - Flask
  - Web requests / data

**Projects**

[Projects Here]
prophecy

# text prediction and generation using Markov Chains
fire alarm is isn't entirely her idea. It spread pearlywhite and she's trying to hurriedly putting my arms are so on to lose her most comfy seats six. The man business is normality. It has? It will take time. It doesn't look tells you with mild invocation the reason behind her son. It was going to give the game seems to be prevented her. It if it's wench this so frightened and so i dart into silence. The hotel toward the steps of ourselves. The sides? It gently holding a meeting his teasing idea. A bat somehow I can't seem too hard. It comes out for one is safe I do you two disconnected things. It sounded and afraid of that. It with a blue while you just lee jordan college and edward we'll see. The games in them. It rang fiona after three times to the happy to acquire in person owns the darkness with dominant pleasure during that possibility of liquids that happens when you think so? The peace and tonight and look ed like we're trying to have a string bean into a matter how long as he grasps lynn's shoulders. The notions amazed that he stands behind one.

Parody generating finished, returning to menu.
```python
from bisect import bisect
import random

def markov_selection(current_word):
    """Return a likely word to follow another word based on frequencies and probability distribution ""
    word_options, frequencies = zip(*current_word)  # decompresses the (word, frequency) tuple
    total = 0
    cumul_frequencies = []
    for frequency in frequencies:
        total += frequency
        cumul_frequencies.append(total)
    randomator = random.random() * total  # random number within the total frequency
    chosen_index = bisect(cumul_frequencies, randomator)  # gets the index of the word to be returned
    return word_options[chosen_index]
```
```python
def predict_three(current_dict, current_dict2):
    '''Get the three most likely words to follow another word'''

    sentence = ''
    full_stop = True

    while True:
        three_keys = []
        word = raw_input('Enter Word: ') # get user input
        sentence = sentence + ' ' + word # add user input to sentence
        if full_stop == True:
            print 'Please try again without a full stop.' # ask again
            word = raw_input('Enter Word: ')
            full_stop = False
        word = word.lower()
        print
        print sentence
        sentence_list = sentence.split() # put user input into a list

        if word not in current_dict:
            three_keys = ['A', 'The', 'It'] # suggest three common words
            print
            print 'Suggested Words'
            print three_keys
            previous_word = word
        else:
            # the first word case, where there is only one word in the list
            if len(sentence_list) == 1:
                predicted = dict(sorted(current_dict[word].items(), key=operator.itemgetter(1), reverse=True)[1:3])
                # returns three highest values
                three_keys.append(predicted.keys())
                previous_word = word
                for key in range(len(three_keys)):
                    if three_keys[key] == 'I':
                        three_keys[key] = 'I'
                        previous_word = word
                        print
                        print 'Suggested Words'
                        print three_keys
            # every other case after the first word
        else:
            two_words = (previous_word, word)
            # create tuple
    ```
From beginning of project until presentation

Had been programming for less than 12 months!

Learned git due to need for collaboration

Became subject of study in English Literature
pathway

# user-generated path-finding web tools using computer vision and the A* algorithm
@app.route('/path_builder', methods=['POST'])
def astart_path():
    """ The following function gets the starting and finishing location from an AJAXian request made from the server, which it then later process through the a_start_search function to return the shortest path. """
    error = "Something Went Wrong"

    if request.method == "POST":
        start_loc = coord_dict[request.form['starting_location']]
        start_loc = start_loc[0]/20, start_loc[1]/20

        finish_loc = coord_dict[request.form['finishing_location']]
        finish_loc = finish_loc[0]/20, finish_loc[1]/20

        path2 = []
        came_from, cost_so_far = a_start_search(diagram4, start_loc, finish_loc)
        path = reconstruct_path(came_from, start_loc, finish_loc)

        for value in path:
            path2.append(((value[1]*20, value[0]*20))

        #path2 = [(0, 0), (0, 200)]
        return jsonify({'data':path2})

    return jsonify({'data':error})
#ray_tracing.py

```python
import cv2
import numpy as np

global height
global width
global channel

filename = 'testmap.png'
img = cv2.imread(filename)

if img is not None:
    print("Image loaded successfully!")

def polygon(x, y):
    """Checks if point (x, y) on the image is within an obstacle or not. Returns true if it is"""
    counter = 0  # keeps track of how many edges have been met
    for i in range(x, width+1):
        colour = (img.item(y, i, 0), img.item(y, i, 1), img.item(y, i, 2))  # keeps track of the current pixel colour
        if img.item(y, i+1, 0), img.item(y, i+1, 1), img.item(y, i+1, 2) != colour:  # if the next pixel is of a different colour,
            counter += 1

    # the parity of the counter determines whether point (x,y) is within an obstacle
    if counter % 2 == 0:  # if even, point (x, y) is not in any obstacles
        return False
    else:  # counter % 2 != 0 and counter != 0:  # if odd, it is
        return True

coordinate_dictionary = {}  # holds tuples of (x, y) values for all traversable nodes with numbered names
wall_list = []  # holds tuples of (x, y) for all untraversable points found by polygon(x, y)
walkable = []  # holds tuples of (x, y) values for all traversable nodes

for i in range(0, width+1, 20):  # these are columns
    for j in range(0, height+1, 20):  # these are rows
        if polygon(i, j) == False:  # if point not in obstacle:
            img[i, j] = [0, 0, 255]
            walkable.append((i, j))
        else:
            wall_list.append((i/20, j/20))

for i in walkable:
    counter += 1
coordinate_dictionary["c"+str(counter)] = i  # numbering/naming the walkable nodes
```

---

**student_showcase**

Bruce Fuda
Highlights

- **4 Weeks+**
  From beginning of project until presentation of 2nd iteration

- **Media Attention**
  Became subject of local media attention

- **Opportunities**
  Approached by local indoor mapping business

student_showcase
Bruce Fuda
k-sum

# monolingual text summarisation using TF-IDF weighting and k-means clustering
Tiger

Tiger (Panthera tigris) and Tigress (disambiguation).

Tigers are apex predators, primarily preying on ungulates such as deer and bovids. They are territorial and generally solitary but social animals, often requiring large contiguous areas of habitat that support their prey. However, introduced tigers in other regions of the world, such as Africa, where they are not a native species, often hybridize with local species.[131] This has led to the loss of genetic diversity in tiger populations. The illegal trade in tiger parts for traditional medicine and the use of tigers in circuses and for hunting trophies and影视 prop has also contributed to the decline of tiger populations.

Conservation status

A Bengal tiger (P. tigris tigris)

A Bengal tiger (P. tigris tigris)

End of Text

Please enter another text: https://en.wikipedia.org/wiki/Tiger
def clusterText(self):
    """This function clusters the sentences based on their TF-IDF scores"""

    # Setting up the KMeans function from sklearn
    n_clusters = number of centroids for points to cluster around. The result from the generateK() function will be used.
    max_iter = Maximum number of iterations the algorithm will run. This is set to 500.
    random_state = The generator used to initialize the centers. If an integer is given, it fixes the seed. Otherwise, the summary will be random every single time.

    Kmeans = KMeans(n_clusters = self.generateK(),
    max_iter=500,
    random_state = 1822)

    # Compute k-means clustering with the TF-IDF scores
    Kmeans.fit(self.tfidf_scores)

    # Will be used to store sentences with their respective cluster.
    clusters = collections.defaultdict(list)

    # Getting what cluster each sentence belongs to.
    linear_matrix = Kmeans.labels_

    # Storing sentence with the respective cluster in a dictionary
    for index, label in enumerate(linear_matrix):
        clusters[label].append(index)

    return clusters

def summary(self):
    """This function gets the cluster with the most items and uses the items to form the summary"""

    # Determining cluster with most items.
    length = 0 # Length of cluster
    values = 0 # Values in the cluster
    for i in dict(self.cluster).values():
        if len(i) > length:
            length = len(i)
            values = i

    # The sentences for the summary are printed.
    for i in values:
        print(self.sentences[i], "\n"
```python
import string

class Summarize:
    def __init__(self, inp):
        self.text = self.detectURL(inp)  # Getting text from input.
        self.sentences = self.getSentence()  # Getting sentences from the text
        self.tfidf_scores = self.getScore()  # Getting TF-IDF scores
        self.cluster = self.clusterText()  # Getting all the clusters of sentences
        self.summary()  # Getting the summary

    def detectURL(self, inp):
        """This function detects if the input is text or URL"""

        # Using regex to check if input is a URL.
        urls = re.findall('http[s]?://(?:[a-zA-Z0-9-][\w.-]*[A-Za-z0-9-])*', inp)

        if urls:
            if there is a URL, getText() function is used to get and return the text
            k = ''.join(urls)
            return self.getText(k)
        else:
            # If there is no URL, the input must have been text. The input is returned.
            return inp

    def getText(self, url):
        """This function takes the URL and extracts the main passage""

        # Getting HTML source of webpage
        page = urllib3.urlopen(url).read().decode('utf8')

        # Extracting relevant text from HTML
        # Source: http://stackoverflow.com/questions/18832567/text-extraction-from-html-data
        soup = BeautifulSoup(page, "html.parser")
        text = ".join(map(lambda p: p.text, soup.find_all('p')))
        return text

    def get_sentence(self):
        """This function takes the text as an input and outputs individual sentences""

        # Splitting chunk of text into sentences
        sentences = sent_tokenize(self.text)
        return sentences
```
Developed based on need of non-native English speakers

Piqued interested in visual translation app
handwriting_recognition

# learned handwriting recognition using a trained neural network
mode = "training"
...
extracts individual letters from grids by finding the gaps
and then scale them to the same size as the input dimensions
...

def prune(data):
    p_con = False
    x = 0  #position of the start of the region (x)
    pruning = []
    for j in range(len(data[0])):  #search all the columns
        con = False
        for i in range(len(data)):  #search each entry in the column
            if data[i][j] == 1:
                #solid grid square was found
                con = True
                if not p_con:
                    #it is the first in the region, mark it
                    x = j
                    break
        #position of the end of the region as found (x)
        #solid squares occurred previously but doesn't now)
        if (p_con and not con) or (j == len(data[0])-1 and p_con):
            p_xcon = False
            y = 0  #position of the start of the region (y)
            for k in range(len(data)):  #search all rows
                xcon = False
                for m in range(x,j):  #search the rows between the start and end of the region's x axis
                    if data[k][m] == 1:
                        #solid grid square was found
                        xcon = True
                        if not p_xcon:
                            #it is the first in the region, mark it
                            y = k
                            break
                if (p_xcon and not xcon) or (k == len(data)-1 and p_xcon):
                    #append the region
                    pruning.append((x,j,y,k))
            p_xcon = xcon
            p_con = con
# sigmoid function
# supports all dimensions of data (scalar (1), vector (nX1), matrix (m,n))
sigmoid = lambda x: 1/(1+np.exp(-x))

# derivative, used for back propagation
sigmoid_der = lambda x: np.exp(-x)/((1+np.exp(-x))**2)

class Neural_Network:
    def __init__(self, i_size, h_size, o_size):
        # parameters for the model of the network
        self.inputSize = i_size  # dimensions of input layer
        self.hiddenSize = h_size  # dimensions of hidden layer
        self.outputSize = o_size  # dimensions of output layer

        ...  
        neural networks are so powerful that extreme data may be remembered rather than
        generalized, this creates horrible predictions. Adding a penalty scalar allows
        the model to generalize more often, creating logical/sensible predictions from
        previous examples
        ...

        self.penalty = 0.8  # penalization scalar of overly complex data (overfitting)

        self.tX = []  # example inputs
        self.tY = []  # example outputs

        ...  
        as each value is passed forward through the network, it is multiplied by a scalar
        known as a weight. These values exist for every possible connection, they are dynamic,
        it is what gives a neural network its mathematical power to model almost everything
        ...

        # initialize random weights for equality of convergence for all desired weights
        self.w1 = np.random.rand(self.inputSize, self.hiddenSize)
        self.w2 = np.random.rand(self.hiddenSize, self.outputSize)

        ...  
        Computes an output for an input
        multiple inputs can be passed at the same time
        ...

        def forward(self, X):
            # sum of all weights + inputs to the hidden layer
            self.z2 = np.dot(X, self.w1)
            # activation function applied to the hidden layer node's input/s
            ...
Highlights

4 Weeks
Including learning about Neural Networks

Confidence
Developed presentation skills and confidence with material

Engaging
Born out of an interest in AI and ML in gaming

Real Challenges
Deeper understanding of efficiency / complexity
spam_away

# spam filtering using a naive Bayesian classifier
Enter the name of the file for classification, or type 'train' to add a file to the training set: myEmail.txt

Most common words are name, email
This is likely to be spam

Enter the name of the file for classification, or type 'train' to add a file to the training set: myEmail2.txt

Most common words are marks, assignment
This is likely to be not spam

Enter the name of the file for classification, or type 'train' to add a file to the training set: myEmail3.txt

Most common words are course, put
This is likely to be not spam

Enter the name of the file for classification, or type 'train' to add a file to the training set: ultimateSpam.txt

Most common words are account, email
This is likely to be spam

Enter the name of the file for classification, or type 'train' to add a file to the training set: train
Enter the name of the spam file to be added: myEmail2.txt

Enter the name of the file for classification, or type 'train' to add a file to the training set: myEmail3.txt

Most common words are course, folder
This is likely to be not spam

Enter the name of the file for classification, or type 'train' to add a file to the training set: }
```python
from functools import reduce
import string
import global evidence dictionary that stores all evidence. tuple[value]
global baseDict dictionary that stores all base rates for classes
global SpamBase #the base rate for spam
global NotSpamBase #the base rate for not spam
global evidenceBaseDict dictionary that stores all base rates for evidence/particular words

#=========================================================================================
#Used to strip words that are common to both spam and non-spam
commonWords = ('the', 'be', 'been', 'has', 'please', 'to', 'of', 'and', 'a', 'in', 'that', 'am', 'no', 'may', 'most', 'due', 'off
#commonWords = () #use this for demonstration #and run myEmail3.txt

#=========================================================================================

#Deal with Training Data
#=========================================================================================
f = open("ultimateSpam.txt", "r") #open the file containing training data

wordcounts = {} #holds the wordcounts for the training data

for word in f.read().split():
    word = word.lower() #lower case everything within the training data
    for c in string.punctuation: #get rid of all punctuation
        word = word.replace(c,"")

    if word not in wordcounts and word not in commonWords: #cut out common words
        wordcounts[word] = 1

    elif word in wordcounts and word not in commonWords:
        wordcounts[word] += 1
```

---

**student_showcase**

Bruce Fuda
#Deal with Data for Classification

test_email = None

while test_email == None:
    email = input("Enter the name of the file for classification, or type 'train' to add a file to the training set: ") #get the test email
    if email != 'train':
        test_email = email #skip the entire else statement
    else:
        #this block will perform a wordcount on the spam file to be added to the training set
        email = input("Enter the name of the spam file to be added: ")
        g = open(email, "r")
        for word in g.read().split():
            word = word.lower() #lower case everything within the training data
            for c in string.punctuation: #get rid of all punctuation
                word = word.replace(c,"")

            if word not in wordcounts and word not in commonWords: #cut out common words
                wordcounts[word] = 1
            elif word in wordcounts and word not in commonWords:
                wordcounts[word] += 1

        g.close()

counter = 1

evidence = {} #a dictionary that holds tuples, in the form (Class, Word):wordcount

while counter < 100:
    for i in wordcounts:
        if wordcounts[i] == counter:
            if wordcounts[i] >= 3: #A word within the training file will be classified as spam if it appears in the training data
                evidence["Spam",i] = wordcounts[i]
            else:
                evidence["Not Spam",i] = wordcounts[i] #if it has less than three occurrences, and is not a common word, it will be placed in the Not Spam list
        counter += 1
Had no prior experience with the concepts

Driven by a desire to learn more about AI

Testing was done with the team working on Prophecy
GASP_simulator

# simulation model for payload recovery from high altitude: Gungahlin Almost Space Project (GASP)
# Configuration File for the G(A)SP Simulator
# Almost all variables should be provided in float form, unless otherwise stated.

# Pygame Screen Variables (Should be integers, otherwise they will be rounded)
width = 800
height = 600

# The Port used to communicate with an external microcontroller (Arduino). MUST be a string.
port = 'COM3'

# Physics Variables
# cd = Co-efficient of Drag (dependent on the shape of the parachute (currently assumed to be circular)).
# area = area of the parachute
# angle = the angle the object faces to start with. (North=90, East=0, South=180, West=270)
# temp = temperature at surface
# glide_angle = tilt of the object while descending
cd = 1.47
mass = 1  # kilograms
area = 1  # m^2
temp = 15.0  # Celsius
slide_angle = 5.0  # degrees

# Wind Variables
wind_scale = 0
wind_direction = -90

# Location Variables
home_lon = 0
home_lat = 0
home_alt = 0

# Object Location Variables. (Where the balloon will start)
longitude = 100
latitude = 100
altitude = 30000
# Imported modules
import pygame
from pygame.locals import *
import math, random
import sys, serial
import gui, physics
import time

from config import *
from gui import HAND_CURSOR # import the second cursor
import googlemaps # used for Google Maps

def update():
    """ Update everything for pygame""
    screen.blit(map_background, (0,0)) # display map background
    pause_button.image = pygame.image.load("images/pause_button.png")
    all_sprites.draw(screen)
    pause_button_group.draw(screen)

    Gui.Variabes(balloon.x, balloon.y, balloon.z,
    theta, balloon.yaw,
    wind_speed, wind_direction) # Send data to the GUI file to update the variables and meters on screen.

    pygame.transform.scale(screen, (0,0))
    int(balloon.x*100), int(balloon.y*100))

    pygame.display.flip()
    pygame.display.update() # Update the screen to show all changes that have been made
    screen.fill((255,255,255)) # fill the spare space where sprites are not located
    clock.tick(60)

class Balloon(pygame.sprite.Sprite):
    """ This class represents the Balloon ""

    def __init__(self, altitude, x, y, home_x, home_y, yaw):
        super(Balloon, self).__init__()
        # balloon image
        self.original_img = pygame.image.load("images/arrow.png")
        self.image = self.original_img
        self.rect = self.original_img.get_rect()
        self.x, self.y = x, y
        self.home_x, self.home_y = home_x, home_y
        self.z = altitude # altitude of the balloon (in meters)
        self.Vx, self.Vy = 1.0, 1.0
Highlights

- **4 Weeks**
  Including physics concepts

- **STEM**
  Designed to work alongside Mechatronics study

- **High visibility**
  Part of a larger project garnering media attention

- **Connectivity**
  Will be used as visual feedback for balloon launch

---

**student_showcase**
Bruce Fuda
why_youShouldCare

# and what you can do to make this less of a big deal
Be positive, active role models

1. Support
   Schools and teachers are looking for technical expertise

2. Advocate
   Ensure others understand the value of a solid grounding in the Digital Technologies / Computer Science

3. Encourage
   Look for ways to get involved with outreach activities such as NCSS Challenge and Young ICT Explorers

4. Enable
   Offer Work Experience and Internship opportunities to kids in high school, not just University

5. Inspire
   Share your successes with the wider community and help create a sense of awe and wonder

Provide the environment that kids need to flourish…
Thanks for your attention

Questions?

bruce.fuda
@Bruce1979
+BruceFuda
bruce@fuda.me
bruce-fuda
http://fuda.me/