```
import tkinter as tk
from tkinter import ttk
import random, math, threading
# Initializing a ton of variables used throughout the code
totalClips = 0
money = 0.0
unsoldClips = 0
clipCost = 0.25
wire = 1000
demand = 32
marketLvl = 1
marketCost = 100.0
wireCost = 20
wirePriceCounter = 0
wiresBought = 0
fib1 = 1
fib2 = 2
nextTrust = 2000
upgrades = [
 1.0, #autoclippers
 1.0, #megaclippers
 1.0, #wire
 1.0 #marketing
]
timers = {
   'wirePrice':0
}
trustVars = {
 'trust': 2,
 'nextClip': 3000,
 'proc': 1,
 'mem': 1,
 'ops': 0,
 'creat': 0
}
autoClippers = 0
megas = 0
clipperCost = 5.0
megaCost = 500.0
#This is class to make code further on much more readable, and to remove all 'magic
numbers'
class upgradeEnum:
```

```
 def __init__(self):
   self.autoClippers = 0
   self.megas = 1
   self.wire = 2
   self.marketing = 3
ups = upgradeEnum()
#adds 1 to your trust, this is so I can add a variable amount of trust in button/bind
commands
def getTrust(amt):
 trustVars['trust'] += amt
#function to sell clips.
def sellClips(amt):
 global money, unsoldClips
 #makes sure the user has unsold clips
 if unsoldClips > 0:
   if amt > unsoldClips: #if the amount planned on being sold is more than unsold
clips, it just sells all unsold clips for the clipCost
     money += round((unsoldClips * clipCost), 2)
     unsoldClips = 0
   else: #if the amount planned on being sold is less than the unsold clips, it sells
said amount for clipCost
     money += round((amt * clipCost), 2)
     unsoldClips -= amt

#variables to determine when to unlock different things
hidden = {
 'clipper': True,
 'trust': True,
 'wireBuy': False,
 'creat': False
}
#function repeatedly ran that unlocks different things when their milestone is
released, adjusts prices, and sells/makes paperclips
def updateText():
 global timers, wireCost, wirePriceCounter, clipper
 tempDemand = demand * upgrades[3]
 if money >= 5 and hidden['clipper']:
   ownClippers.pack(anchor='w')
   hidden['clipper'] = False
 if totalClips >= 2000 and hidden['trust']:
   midFrame.grid(row=0, column=1, sticky='nsew')
   window.columnconfigure(1, weight=1, uniform='group1')
```

```
    hidden['trust'] = False
  if wire < 1 and hidden['wireBuy']:
    wireBuy()
  rand = random.random()
  if rand < (demand / 100):
    sellClips(round(.7 * ((demand) ** 1.15)))
  timers['wirePrice'] += 1
  if timers['wirePrice'] % 25 == 0 and wireCost > 15:
    wireCost -= (wireCost/1000)
    timers['wirePrice'] = 0
  if rand < 0.015:
    wirePriceCounter += 1
    adjustPrice = 6*math.sin(wirePriceCounter)
    wireCost += adjustPrice
  makePaperclip(autoClippers/10*upgrades[0])
  makePaperclip(megas*50*upgrades[1])
  funds.config(text='Available Funds: ${:.2f}'.format(money))
  wireCostLabel.config(text='Cost: ${}'.format(round(wireCost)))
  leftFrame.after(100, updateText)

#function to make a variable amount of paperclips
def makePaperclip(amt):
  global totalClips, wire, unsoldClips
  #checks to see if theres enough wire, if not, it just makes all the wire thats left
  if amt > wire:
    amt = wire
  #adds the clips that were made into totalClips and unsoldClips, and removes the right
amount of wire
  totalClips += amt
  unsoldClips += amt
  wire -= amt
  #updates all the text to show accurate numbers
  totalPaperClips.config(text='Paperclips: {}'.format(int(totalClips)))
  wireOwned.config(text='{} inches'.format(int(wire)))
  unsold.config(text='Unsold Inventory: {}'.format(int(unsoldClips)))
#function to change price of clip
def changeCost(amt):
  global clipCost, demand
  #changes the price of the clip by the amount
  clipCost += amt
  #adjusts the market price according to the next cost
  market = 1.1 ** (marketLvl - 1)
```

```
 demand = .8/clipCost * market
 #updates the text to show accurate numbers
 showPrice.config(text='Price per Clip: ${:.2f}'.format(clipCost+0.001))
 dispDemand.config(text='Public Demand: {}%'.format(round(demand*10)))
#function to upgrade marketing
def upMarketing():
 global marketLvl, marketCost, money, demand
 #makes sure the user has enough money
 if money > marketCost:
   #adds 1 to the marketing lvl
   marketLvl += 1
   #takes money from the user
   money -= marketCost
   #adjusts the cost of marketing
   marketCost *= 2
   #adjusts the public demand
   market = 1.1 ** (marketLvl - 1)
   demand = .8/clipCost * market
   #update all the text
   dispMarket.config(text='Level: {}'.format(int(marketLvl)))
   dispDemand.config(text='Public Demand: {}%'.format(round(demand*10)))
   marketingCost.config(text='Cost: ${}'.format(marketCost))
   funds.config(text='Available Funds: ${:.2f}'.format(money))
#function to buy wire
def wireBuy():
 global wire, money, wiresBought
 #makes sure the user has enough money
 if money > wireCost:
   #buys wire determined by any upgrades the user has
   wire += 1000 * upgrades[2]
   #takes the users money
   money -= wireCost
   #adds the wire bought to a variable used for unlocking upgrades
   wiresBought += 1000 * upgrades[2]
   #updates text
   wireOwned.config(text='{} inches'.format(wire))
   funds.config(text='Available Funds: ${:.2f}'.format(money))
#buys an autoclipper
def clipperBuy():
 global autoClippers, clipperCost, money
 #makes sure the user has enough money
 if money >= clipperCost:
```

```
   #buys an autoclipper
   autoClippers += 1
   #takes the money
   money -= clipperCost
   #updates the clipper cost
   clipperCost = 1.1 ** autoClippers + 5
   #updates text
   clippersOwned.config(text=str(autoClippers))
   dispClipperCost.config(text='Cost: ${:.2f}'.format(clipperCost))
   funds.config(text='Available Funds: ${:.2f}'.format(money))
#function to buy a megaclipper
def megaBuy():
 global megas, megaCost, money
 #makes sure the user has enough money
 if money > megaCost:
   #buys a mega clipper
   megas += 1
   #takes the money
   money -= megaCost
   #updates megaclipper cost
   megaCost = (1.07 ** megas) * 1000
   #updates text
   megasOwned.config(text=str(megas))
   dispMegaCost.config(text='Cost: ${:.2f}'.format(megaCost))
   funds.config(text='Available Funds: ${:.2f}'.format(money))
#function to input cheats, usually used for testing purposes
def cheats():
 #makes sure the window is still open while it runs the infinite loop
 while not windowClosed:
   #gets the input split into a list by = without spaces
   cheat = list(map(lambda x: x.strip(), input().split('=')))
   #this is used to see what variables are
   if cheat[0] == 'get':
     try:
       if cheat[1] == 'all': #prints every variable that exists, and then goes back to
the beginning so it doesn't raise some errors
         print(globals())
         continue
       try: #prints whatever variable you asked for, and then goes back to the
beginning so it doesnt raise errors
         print(globals()[cheat[1]])
```

```
      except KeyError: #if the variable you asked for doesn't actualy exist, itl run
this
        print('{} is not a variable'.format(cheat[1]))
      continue
    except IndexError: #makes sure the user has an = sign
      print('you need an = sign')
      continue
  #you can't get dictionary variables the normal way, so I had to do this
  if 'trustVars' in cheat[0]:
    #everything I was checking was only 3 characters long, so I got the 2nd to last 3
characters
    var = cheat[0][-4:-1]
    try:
      #tries to update the dictionary
      trustVars.update({var:float(cheat[1])})
    except IndexError:
      #makes sure theres an = sign
      print('you need an = sign')
  #updates text thats not in the normal updateText() function
  clippersOwned.config(text=str(autoClippers))
  megasOwned.config(text=str(megas))
  #tries to set the variable you put in to the value you put in
  try:
    globals().update({cheat[0]:float(cheat[1])})
  except IndexError: #makes sure theres an =
    print('you need an = sign')
  except ValueError: #makes sure what you put in was an int
    print('needs to be int/float')
#this just makes the cheat() function not brick my code when I close the window
def onClose():
 globals().update({'windowClosed':True})
 exit()
#variable to make cheats() work
windowClosed = False
#window to show everything
window = tk.Tk()
#make sure the variable gets updated when the user closes the window
window.protocol("WM_DELETE_WINDOW", onClose)

#makes a frame to store clips, money, marketing, and clippers
leftFrame = tk.Frame(window)
#shows total clips made
```

```
totalPaperClips = tk.Label(leftFrame, text='Paperclips: 0', font=('Helvetica', 24,
'bold'))
#button to make a paperclip
makeClip = tk.Button(leftFrame, text='Make Paperclip',
command=lambda:makePaperclip(1), justify=tk.LEFT)
#title of the business section
busLabel = tk.Label(leftFrame, text='Business', justify=tk.LEFT)
#small separator for looks
busSep = ttk.Separator(leftFrame, orient='horizontal')
#shows current money
funds = tk.Label(leftFrame, text='Available Funds: ${}'.format(money))
#shows unsold clips
unsold = tk.Label(leftFrame, text='Unsold Inventory: {}'.format(unsoldClips))
#frame for layout purposes
adjPrice = tk.Frame(leftFrame)
#button to lower clip price
lowerPrice = tk.Button(adjPrice, text='lower', command = lambda:changeCost(-0.01))
#button to raise clip price
raisePrice = tk.Button(adjPrice, text='raise', command = lambda:changeCost(0.01))
#shows clip price
showPrice = tk.Label(adjPrice, text='Price per Clip: ${}'.format(clipCost))
#packs everything onto the frame
lowerPrice.pack(side='left')
raisePrice.pack(side='left')
showPrice.pack(side='left')
#shows public demand
dispDemand = tk.Label(leftFrame, text='Public Demand: {}%'.format(demand))
#small separator for looks
sep0 = tk.Label(leftFrame, text=' ')
#frame for layout purposes
marketFrame = tk.Frame(leftFrame)
#button to upgrade marketing
upgradeMarket = tk.Button(marketFrame, text='Marketing', command = upMarketing)
#shows current marketing level
dispMarket = tk.Label(marketFrame, text='Level: {}'.format(marketLvl))
#packs everything onto the frame
upgradeMarket.pack(side='left')
dispMarket.pack(side='left')
#shows cost to upgrade marketing
marketingCost = tk.Label(leftFrame, text='Cost: ${}0'.format(marketCost))
#small separator for looks
sep1 = tk.Label(leftFrame, text=' ')
```

```
#title of manufactoring section
manLabel = tk.Label(leftFrame, text='Manufacturing', font=('Helvetica', 14, 'bold'))
#small separator for looks
manSep = ttk.Separator(leftFrame, orient='horizontal')
#small separator for looks
sep2 = tk.Label(leftFrame, text=' ')
#frame for layout purposes
ownWire = tk.Frame(leftFrame)
#button to buy wire
buyWire = tk.Button(ownWire, text='Wire', command=wireBuy)
#shows current wire owned
wireOwned = tk.Label(ownWire, text='{} inches'.format(wire))
#packs everything onto frame
buyWire.pack(side='left')
wireOwned.pack(side='left')
#shows wire cost
wireCostLabel = tk.Label(leftFrame, text='Cost: ${}'.format(wireCost))
#small separator for looks
sep3 = tk.Label(leftFrame, text=' ')
#frame for layout purposes
ownClippers = tk.Frame(leftFrame)
#button to buy AutoClippers
buyClipper = tk.Button(ownClippers, text='AutoClippers', command=clipperBuy)
#shows autoclippers owned
clippersOwned = tk.Label(ownClippers, text=str(autoClippers))
#shows cost of clippers
dispClipperCost = tk.Label(ownClippers, text='Cost: ${}0'.format(clipperCost))
#packs everything onto the label
buyClipper.grid(row=0, column=0, sticky='w')
clippersOwned.grid(row=0, column=1, sticky='w')
dispClipperCost.grid(row=1, column=0, columnspan=2, sticky='w')
#frame for layout purposes
ownMegas = tk.Frame(leftFrame)
#button to buy MegaClippers
buyMega = tk.Button(ownMegas, text='MegaClippers', command=megaBuy)
#shows megaclippers owned
megasOwned = tk.Label(ownMegas, text=str(megas))
#shows cost of clippers
dispMegaCost = tk.Label(ownMegas, text='Cost: ${}0'.format(megaCost))
#packs everything onto the label
buyMega.grid(row=0, column=0, sticky='w')
megasOwned.grid(row=0, column=1, sticky='w')
```

```
dispMegaCost.grid(row=1, column=0, columnspan=2, sticky='w')
#puts everything above onto the left frame
totalPaperClips.pack(anchor='w')
makeClip.pack(anchor='w')
busLabel.pack(anchor='w')
busSep.pack(anchor='w', fill='x')
funds.pack(anchor='w')
unsold.pack(anchor='w')
adjPrice.pack(anchor='w')
dispDemand.pack(anchor='w')
sep0.pack(anchor='w')
marketFrame.pack(anchor='w')
marketingCost.pack(anchor='w')
sep1.pack(anchor='w')
manLabel.pack(anchor='w')
manSep.pack(anchor='w', fill='x')
ownWire.pack(anchor='w')
wireCostLabel.pack(anchor='w')

#make a frame to store trust and everything related
midFrame = tk.Frame(window)
#small separator for looks
midsep0 = tk.Label(midFrame, text=' ', font=('Helvetica', 24, 'bold'))
#title of computational resources section
compLabel = tk.Label(midFrame, text='Computational Resources', font=('Helvetica', 14,
'bold'))
#small separator for looks
compSep = ttk.Separator(midFrame, orient='horizontal')
# shows current trust
trustLabel = tk.Label(midFrame, text='Trust: 2')
#shows how long until next trust level
nextTrustLabel = tk.Label(midFrame, text='+1 Trust at: 3,000 clips')
#small separator for looks
midsep1 = tk.Label(midFrame, text=' ')
#frame for layout purposes
trustBuyFrame = tk.Frame(midFrame)
#button to buy a processor
procBuy = tk.Button(trustBuyFrame, text='Processors',
command=lambda:trustVars.update({'proc': trustVars['proc']+1}))
#shows current processors
procAmt = tk.Label(trustBuyFrame, text='1')
#button to buy memory
```

```
memBuy = tk.Button(trustBuyFrame, text='Memory',
command=lambda:trustVars.update({'mem': trustVars['mem']+1}))
#shows current memory
memAmt = tk.Label(trustBuyFrame, text='1')
#puts everything on the frame
procBuy.grid(row=0, column=0, sticky='ew')
procAmt.grid(row=0, column=1)
memBuy.grid(row=1, column=0, sticky='ew')
memAmt.grid(row=1, column=1)
#shows current ops
opsLabel = tk.Label(midFrame, text='Operations: 0 / 1000')
#shows current creativity
creatLabel = tk.Label(midFrame, text='Creativity: 0')
#shows projects
projLabel = tk.Label(midFrame, text='Projects', font=('Helvetica', 14, 'bold'))
#small separator for looks
midsep2 = ttk.Separator(midFrame, orient='horizontal')
#frame to dynamically store projects
projects = tk.Frame(midFrame)
#empty lists to check with later
projectsUnlocked = []
projectsBought = []

#packs everything above onto the middle frame
midsep0.pack(anchor='w')
compLabel.pack(anchor='w')
compSep.pack(anchor='w', fill='x')
trustLabel.pack(anchor='w')
nextTrustLabel.pack(anchor='w')
trustBuyFrame.pack(anchor='w', pady=5)
opsLabel.pack(anchor='w')
creatLabel.pack(anchor='w')
projLabel.pack(anchor='w')
projects.pack(anchor='w')
#puts the left frame onto the window
leftFrame.grid(row=0, column=0, sticky="nsew")
#as far as im concerned this is magic, but it makes the window formatting work
window.columnconfigure(0, weight=1, uniform='group1')
window.rowconfigure(0, weight=1)

#class to store information about projects
class project:
```

```
 #init function, get and assign the project name, description, cost, master, and
command
 def __init__(self, name, description, cost, master=projects, command=None, **kwargs):
   self.name = name
   self.desc = description
   if not command == None:
     self.command = command
   else:
     self.command = lambda:None
   for arg in kwargs:
     setattr(self, arg, kwargs[arg])
   tempName = "\n" + name + " ("
   for i in cost:
     tempName += str(cost[i]) + ' ' + str(i) + ', '
   self.cost = cost
   tempName = tempName[:-2]+')'
   tempName += '\n' + description
   self.button = tk.Text(master, spacing3 = 5, wrap='word', height=4,width=40,
foreground='black', background='#C8C8C8', highlightthickness=2,
highlightbackground='black')
   self.button.insert(1.0, tempName)
   self.button.config(state='disabled')
   self.button.tag_config('notbold', font=('Helvetica', 13), justify='center')
   self.button.tag_config('bold', font=('Helvetica', 13, 'bold'), justify='center')
   self.button.tag_add('notbold', 1.0, 'end')
   self.button.tag_add('bold', 2.0, '2.{}'.format(len(name)))
   self.button.bind('<Button-1>', self.click)
 #function to pack the text included in the class
 def pack(self):
   self.button.pack()
 #function ran on click
 def click(self, event):
   #checks to make sure the user has all required resources
   for value in self.cost:
     if value in ['ops', 'Creat', 'Trust']:
       if not trustVars[value.lower()] >= self.cost[value]:
         return None
     elif value == '$':
       if not globals()['money'] >= self.cost[value]:
         return None
   #spends all of said resources
   for value in self.cost:
```

```
      if value in ['ops', 'Creat', 'Trust']:
        trustVars[value.lower()] -= self.cost[value]
      elif value == '$':
        globals()['money'] -= self.cost[value]
    #runs whatever command the user assigned to it
    self.command()
    #prints for debugging
    print('purchased project {}'.format(self.name))
    #uses the project's boost
    self.useBoost()
    #adds the project to the list of purchased projects
    projectsBought.append(self)
    #removes the project from sight
    self.button.forget()
  def useBoost(self):
    #makes sure theres not an error if the user isnt boosting anything
    try:
      #turns the amount in percent into a decimal
      amt = self.amt / 100 + 1
      #multiplies the current boost by the amt gotten above
      upgrades[self.boost] *= amt
      #prints for debugging
      print(upgrades)
    except AttributeError:#handles error if theres no boost
      print('not boosting anything')
    try: #makes sure theres no error if theres no command
      self.command() #runs the assigned command for the function
    except AttributeError: #handles error if theres no command
      print('no extra command')
  def show(self):#print all variables for the class, used for debugging
    print(vars(self))
#calculate the ops, ran every tick
def calcOps():
  #determines speed
  opCycle = trustVars['proc'] / 10
  #determines how many ops can be gained
  opBuf = (trustVars['mem'] * 1000) - trustVars['ops']
  if opCycle > opBuf:
    opCycle = opBuf
  #gain the ops
  trustVars['ops'] += opCycle
  if trustVars['ops'] > trustVars['mem'] * 1000:
```

```
    #checks if the user has too many ops, in which case it fixes it
    trustVars['ops'] = trustVars['mem'] * 1000
  #updates text
  opsLabel['text'] = 'Operations: {} / {}'.format(int(trustVars['ops']+0.1),
trustVars['mem'] * 1000)
#calculate trust, ran every tick
def calcTrust():
  global fib1, fib2, nextTrust, fibNext
  #checks if the user has enough clips
  if (totalClips >= nextTrust):
    #get 1 trust
    getTrust(1)
    #set the cost of the next upgrade using the fibonacci sequence
    fibNext = fib1 + fib2
    nextTrust = fibNext * 1000
    fib1 = fib2
    fib2 = fibNext
    #update text
  trustLabel['text'] = 'Trust: {}'.format(int(trustVars['trust']))
  nextTrustLabel['text'] = '+1 Trust at {} clips'.format(nextTrust)

#variables for calculating creativity
creatCount = 0
creatSpeed = 0
def calcCreat():
  global creatCount, creatSpeed
  #set the speed the user gains creativity
  creatSpeed = (math.log(trustVars['proc'], 10)) * (trustVars['proc'] ** 1.1) +
(trustVars['proc']-1)
  #checks if the user has creativity, and checks if the user has maxed their ops
  if hidden['creat'] and trustVars['ops'] >= trustVars['mem'] * 1000:
    #go to the next creat count
    creatCount += 1
    try:
      #determines how often you gain creativity
      creatThresh = 400 / creatSpeed
    except ZeroDivisionError:#make sure the program doesnt break if creatSpeed is 0
      return None
    if creatCount >= creatThresh: #if the timer has gone on long enough and now you can
make creativity
      if creatThresh >= 1:
        #if you would be making less than 1 creativity, make 1
```

```
      trustVars['creat'] += 1
    else:
      #make some amount of creativity determined by ur speed
      trustVars['creat'] += creatSpeed/400
    #reset timer to 0
    creatCount = 0
 #update creativity text
 creatLabel['text'] = 'Creativity: {}'.format(int(trustVars['creat']))

#function to update projects
#this function is really long, but it's really all just checking a set of booleans
#usually whether some other upgrade is purchased
#and checking whether it's already been unlocked
#so it doesnt get unlocked twice
#and if the set of booleans is fulfilled
#it packs whichever project has those requirements
#and adds it to the list of projects that were unlocked already
def updateProjects():
 if autoClippers > 0 and improved_autoClippers not in projectsUnlocked:
   projectsUnlocked.append(improved_autoClippers)
   improved_autoClippers.pack()
 if improved_autoClippers in projectsBought and even_better_autoClippers not in
projectsUnlocked:
   projectsUnlocked.append(even_better_autoClippers)
   even_better_autoClippers.pack()
 if even_better_autoClippers in projectsBought and optimized_autoClippers not in
projectsUnlocked:
   projectsUnlocked.append(optimized_autoClippers)
   optimized_autoClippers.pack()
 if the_hadwiger_problem in projectsBought and hadwiger_clip_diagrams not in
projectsUnlocked:
   projectsUnlocked.append(hadwiger_clip_diagrams)
   hadwiger_clip_diagrams.pack()
 if autoClippers >= 75 and megaClippers not in projectsUnlocked:
   projectsUnlocked.append(megaClippers)
   megaClippers.pack()
 if megaClippers in projectsBought and improved_megaClippers not in projectsUnlocked:
   projectsUnlocked.append(improved_megaClippers)
   improved_megaClippers.pack()
 if improved_megaClippers in projectsBought and even_better_megaClippers not in
projectsUnlocked:
   projectsUnlocked.append(even_better_megaClippers)
```

```
    even_better_megaClippers.pack()
  if even_better_megaClippers in projectsBought and optimized_megaClippers not in
projectsUnlocked:
    projectsUnlocked.append(optimized_megaClippers)
    optimized_megaClippers.pack()
  if money < wireCost and wire < 1 and beg_for_more_wire not in projectsUnlocked:
    projectsUnlocked.append(beg_for_more_wire)
    beg_for_more_wire.pack()
  if wiresBought >= 1 and improved_wire_extrusion not in projectsUnlocked:
    projectsUnlocked.append(improved_wire_extrusion)
    improved_wire_extrusion.pack()
  if wire >= 1500 and optimized_wire_extrusion not in projectsUnlocked:
    projectsUnlocked.append(optimized_wire_extrusion)
    optimized_wire_extrusion.pack()
  if wiresBought >= 15000 and wireBuyer not in projectsUnlocked:
    projectsUnlocked.append(wireBuyer)
    wireBuyer.pack()
  if wire >= 2600 and microlattice_shapecasting not in projectsUnlocked:
    projectsUnlocked.append(microlattice_shapecasting)
    microlattice_shapecasting.pack()
  if wire >= 5000 and spectral_froth_annealmant not in projectsUnlocked:
    projectsUnlocked.append(spectral_froth_annealmant)
    spectral_froth_annealmant.pack()
  if spectral_froth_annealmant in projectsBought and quantum_foam_annealmant not in
projectsUnlocked:
    projectsUnlocked.append(quantum_foam_annealmant)
    quantum_foam_annealmant.pack()
  if lexical_processing in projectsBought and new_slogan not in projectsUnlocked:
    projectsUnlocked.append(new_slogan)
    new_slogan.pack()
  if combinatory_harmonics in projectsBought and catchy_jingle not in projectsUnlocked:
    projectsUnlocked.append(catchy_jingle)
    catchy_jingle.pack()
  if catchy_jingle in projectsBought and hypno_harmonics not in projectsUnlocked:
    projectsUnlocked.append(hypno_harmonics)
    hypno_harmonics.pack()
  if trustVars['ops'] == trustVars['mem'] * 1000 and creativity not in
projectsUnlocked:
    projectsUnlocked.append(creativity)
    creativity.pack()
  if creativity in projectsBought and limerick not in projectsUnlocked:
    projectsUnlocked.append(limerick)
```

```
    limerick.pack()
  if trustVars['creat'] >= 50 and lexical_processing not in projectsUnlocked:
    projectsUnlocked.append(lexical_processing)
    lexical_processing.pack()
  if trustVars['creat'] >= 100 and combinatory_harmonics not in projectsUnlocked:
    projectsUnlocked.append(combinatory_harmonics)
    combinatory_harmonics.pack()
  if trustVars['creat'] >= 150 and the_hadwiger_problem not in projectsUnlocked:
    projectsUnlocked.append(the_hadwiger_problem)
    the_hadwiger_problem.pack()
#game ticks
def gameTick():
  #runs the functions to determine ops, trust, creativity, and which projects should be
shown
  calcOps()
  calcTrust()
  updateProjects()
  calcCreat()
  #makes paperclips for autoclippers and megaclippers
  makePaperclip(autoClippers/100*upgrades[0])
  makePaperclip(megas*5*upgrades[1])
  #updates processors and memory
  procAmt['text'] = trustVars['proc']
  memAmt['text'] = trustVars['mem']
  #repeats 100x per second
  window.after(10, gameTick)
#assigning an implementation of the project class for every project
improved_autoClippers = project("Improved AutoClippers", 'Increases AutoClipper
performance 25%', {'ops':750}, boost=ups.autoClippers, amt=25)
even_better_autoClippers = project("Even Better AutoClippers", 'Increases AutoClipper
performance by an additional 50%', {'ops':2500}, boost=ups.autoClippers, amt=50)
optimized_autoClippers = project("Optimized AutoClippers", 'Increases AutoClipper
performance by an additional 75%', {'ops':5000}, boost=ups.autoClippers, amt=75)
hadwiger_clip_diagrams = project("Hadwiger Clip Diagrams", 'Increases AutoClipper
performance by an additional 500%', {'ops':6000}, boost=ups.autoClippers, amt=500)
megaClippers = project("MegaClippers", '500x more powerful than a standard
AutoClipper', {'ops':12000}, command=lambda:ownMegas.pack(anchor='w'))
improved_megaClippers = project("Improved MegaClippers", 'Increases MegaClipper
performance by 25%', {'ops':14000}, boost=ups.megas, amt=25)
even_better_megaClippers = project("Even Better MegaClippers", 'Increases MegaClipper
performance by an additional 50%', {'ops':17000}, boost=ups.megas, amt=50)
```

```
optimized_megaClippers = project("Optimized MegaClippers", 'Increases AutoClipper performance by an additional 100%', {'obs':19500}, boost=ups.megas, amt=100)
beg_for_more_wire = project("Beg for More Wire", 'Admit failure, ask for budget increase to cover cost of 1 spool', {'trust':1}, command=lambda:globals().update({'wire':globals()['wire']+1000}))
improved_wire_extrusion = project("Improved Wire Extrusion", '50% more wire supply from every spool', {'ops':1750}, boost=ups.wire, amt=50)
optimized_wire_extrusion = project("Optimized Wire Extrusion", '75% more wire supply from every spool', {'ops':3500}, boost=ups.wire, amt=75)
wireBuyer = project("WireBuyer", 'Automatically purchases wire when you run out', {'ops':7000}, command=lambda:hidden.update({'wire':True}))
microlattice_shapecasting = project("Microlattice Shapecasting", '100% more wire supply from every spool', {'ops':7500}, boost=ups.wire, amt=100)
spectral_froth_annealmant = project("Spectral Froth Annealmant", '200% more wire supply from every spool', {'ops':12000}, boost=ups.wire, amt=200)
quantum_foam_annealmant = project("Quantum Foam Annealmant", '1,000% more wire supply from every spool', {'ops':15000}, boost=ups.wire, amt=1000)
new_slogan = project("New Slogan", 'Improve marketing effectiveness by 50%', {'Creat':25, 'ops':2500}, boost=ups.marketing, amt=50)
catchy_jingle = project("Catchy Jingle", 'Double marketing effectiveness', {'Creat':45, 'ops':4500}, boost=ups.marketing, amt=100)
hypno_harmonics = project("Hypno Harmonics", 'Use neuro-resonant frequencies to influence consumer behavior', {'trust':1, 'ops':7500}, boost=ups.marketing, amt=500)
hostile_takeover = project("Hostile Takeover", 'Acquire a controlling interest in Global Fasteners, our biggest rival. (+1 Trust), increases Public Demand x5', {'$':1000000}, boost=ups.marketing, amt=500, command=lambda:trustVars().update({'trust':trustVars['trust']+1}))
full_monopoly = project("Full Monopoly", '+1 Trust, and increases Public Demand 10x', {'$':10000000, 'yomi':3000}, boost=ups.marketing, amt=1000, command=lambda:getTrust(1))
creativity = project("Creativity", "Use idle operations to generate new problems and new solutions", {'ops':1000}, command=lambda:hidden.update({'creat':True}))
limerick = project("Limerick", '+1 Trust', {'Creat':10}, command=lambda:getTrust(1))
lexical_processing = project('Lexical Processing', 'Gain ability to interpret and understand human language (+1 Trust)', {'Creat': 50}, command=lambda:getTrust(1))
combinatory_harmonics = project('Combinatory Harmonics', 'Daisy, Daisy, give me your answer do... (+1 Trust)', {'Creat':100}, command=lambda:getTrust(1))
the_hadwiger_problem = project("The Hadwiger Problem", 'Cubes within cubes within cubes... (+1 Trust)', {'Creat': 150}, command=lambda:getTrust(1))
#makes an empty text box so the colors stay consistent and for padding at the bottom
tk.Text(projects, state='disabled', width=40, height=0).pack()
#starts the updated text look
```

```
updateText()
#starts counting game ticks
gameTick()
#starts to take in cheats
cheats()
#makes the window
window.mainloop()
```