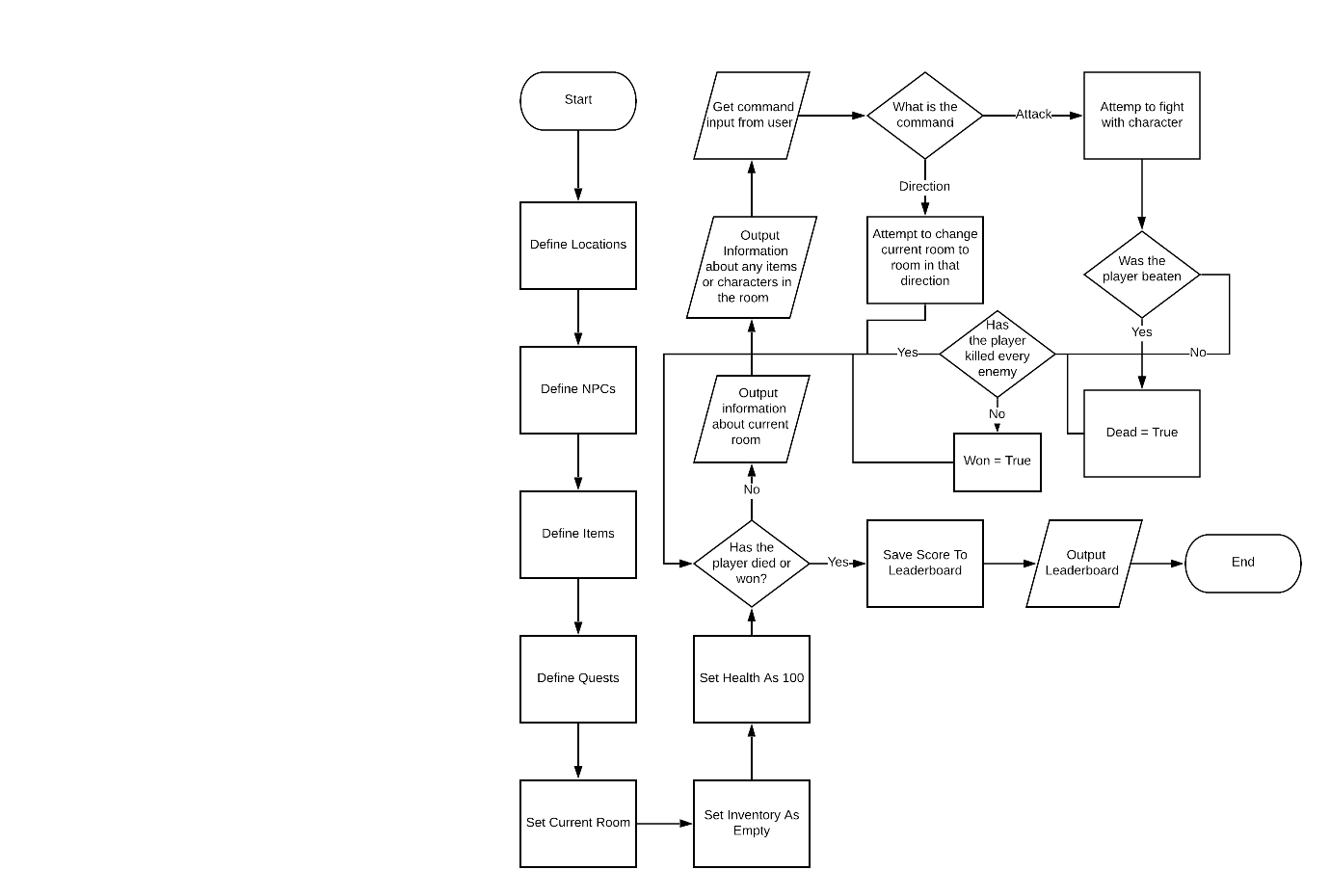
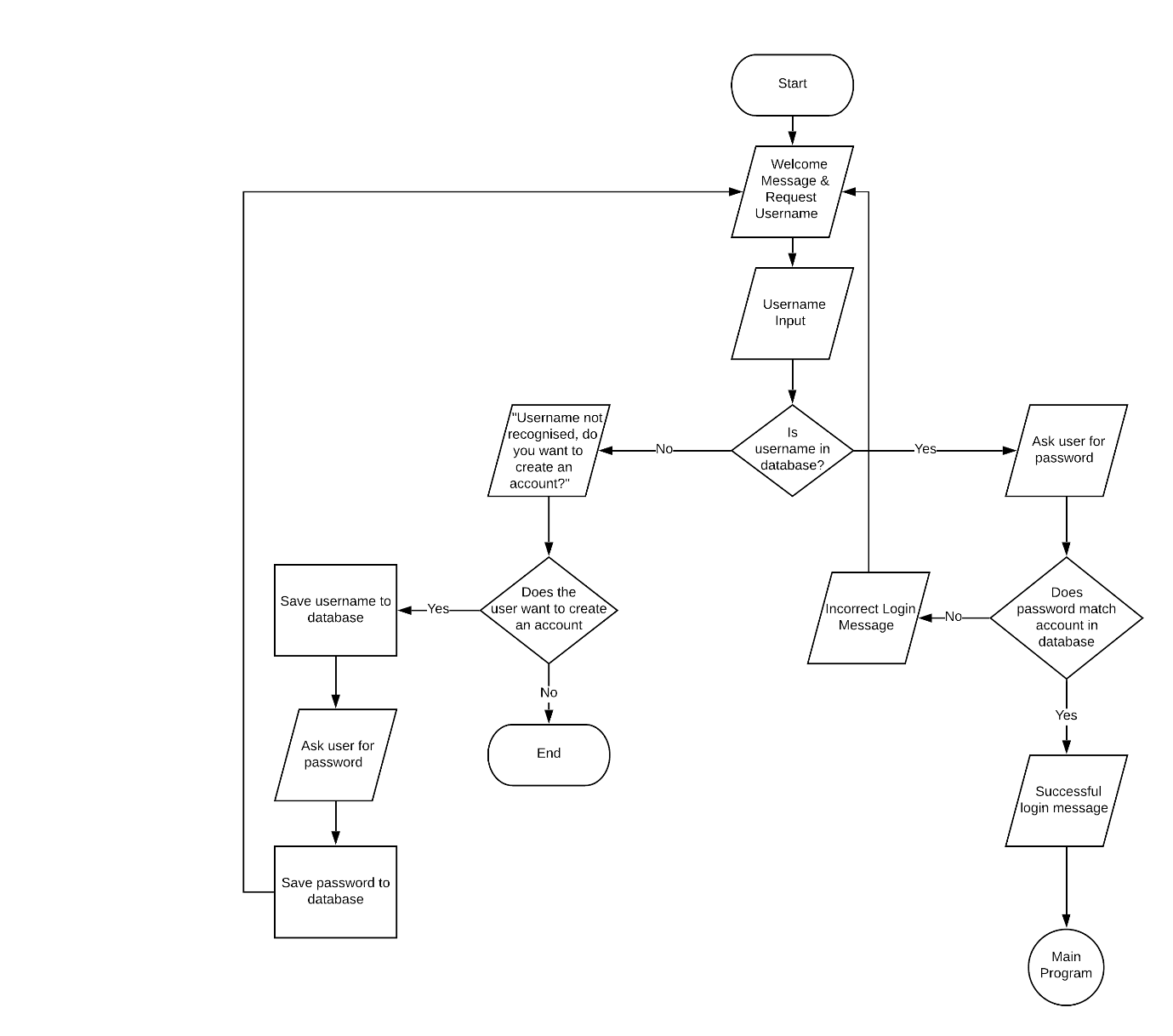
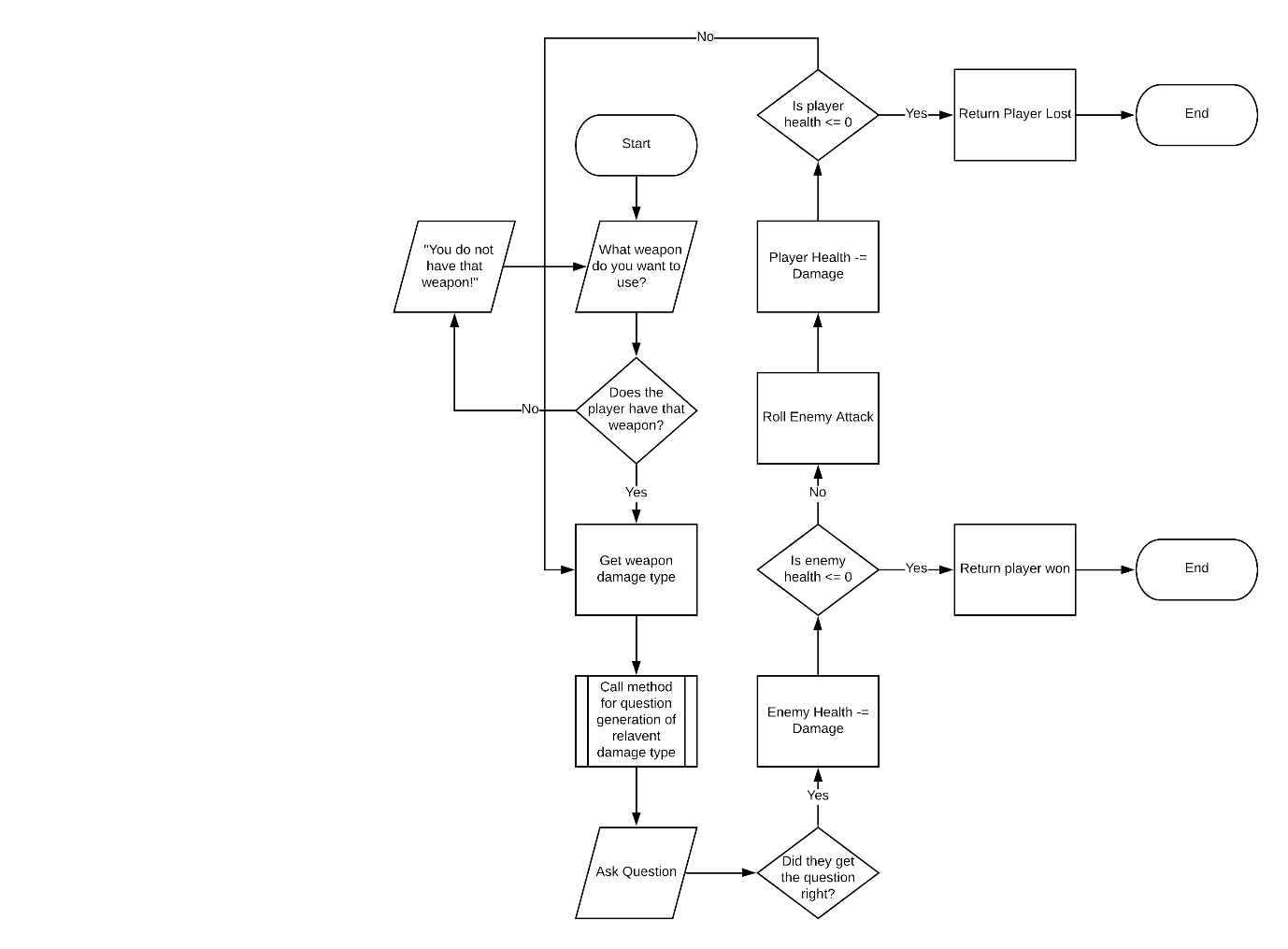
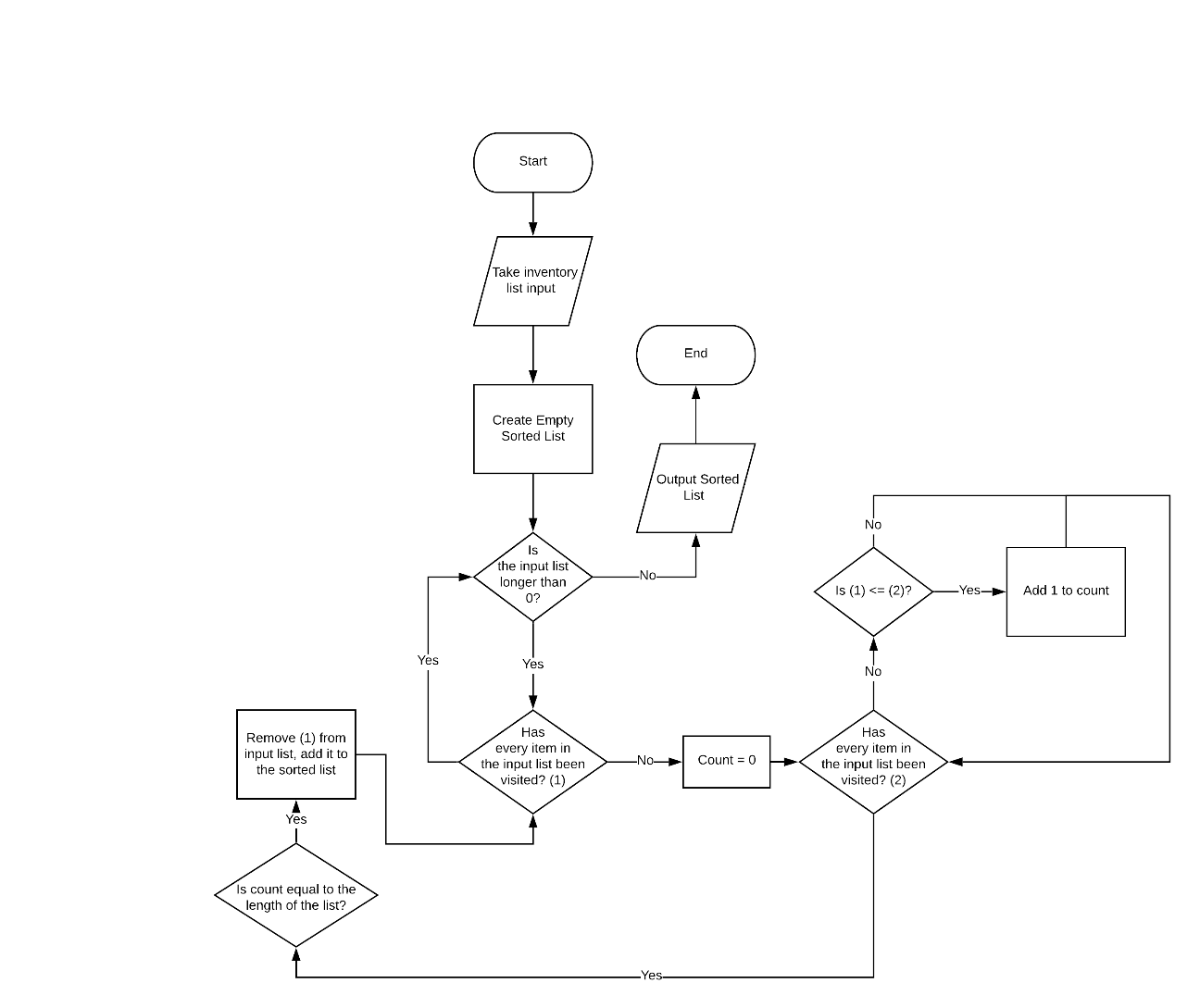
NEA Analysis

* Introduction
  + Text adventure games are a form of interactive literature used to convey a story through customisable options that make it a game. They are the origins of lots of computer games today, since they could run in the basic consoles of old computers. The first text adventure game was Colossal Cave Adventure in 1976, and is still available to play an updated version at <https://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure>
  + Educational games are also common in teaching, as they can make often boring learning fun which helps students learn
  + My target audience is Yr12 Computer Science Students, and my client is a Computer Science Teacher. Yr12 are old enough to grasp complex ideas so the game can have some complex mechanics. However, it does require more complex gameplay/questions to engage he audience.
  + A common design amongst existing text-adventures is that they can understand full sentences and convert them into commands, for example rather than saying “attack npc1” you can say “stab ncp1 in the chest” and it will know that it should be attacking npc1.
* Plan for Weapon Class Attributes:
  + Item sub class
  + Damage type, controls what type of questions to give and relates to the enemy weaknesses
  + Value, for trading with merchants
  + Name
  + Weapon type. E.g sword, bow, ect.







Program Code:

Main.py

from textadventure import \* #Imports Text Adventure Modules

import sys,time,getpass,random #Imports Other Required Modules

def type(string): #Creates Function For Outputting Letter By Letter

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.02)

sys.stdout.write("\n")

sys.stdout.flush()

def type2(string): #Same As Above Function Just Slower

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.04)

sys.stdout.write("\n")

sys.stdout.flush()

file = open("data.txt","a+") #Creates Data.txt If It Doesn't Already Exist

file.close()

type("Welcome to the text adventure what is your username?") #Runs Login/Account Creation

username = input("> ")

found= False

x = 0

file = open("data.txt","r")

database = []

for line in file:

line = line.split("kyuip")

if line[0].lower() == username.lower():

found = True

location = x

database.append([line[0],line[1]])

x += 1

file.close()

if found:

type("What is your password?")

password = getpass.getpass("> ")

if password == database[location][1].rstrip():

type("Welcome To The Text adventure")

else:

type("Invalid Login")

exit()

else:

type("Username not in database, do you want to create an account? (Y/N)")

answer = input("> ")

if answer.lower() == "y":

type("What do you want your password to be?")

password = getpass.getpass("> ")

type("Please Confirm Password")

password2 = getpass.getpass("> ")

if password == password2:

file = open("data.txt","a")

file.write(username+"kyuip"+password+"\n")

else:

type("Passwords Do Not Match")

exit()

else:

exit()

class Player: #Creates Class To House Player Attriubtes

def \_\_init\_\_(self):

self.health = 100 #The Health Of The Player, Max 100

self.inventory = {} #The Dictionary Of The Player's Items

self.equipedWeapon = None #The Currently Equipped Weapon

self.dead = False #Whether Or Not The Player Is Dead

self.victorious = False #Whether Or Not The Player Has Defeated The Final Boss

self.location = None #The Player's Current Locations

self.score = 0 #The Player's Score

self.cash = 0 #The Player's Money

def move(self,location): #Method For Chaning Location Attribute

self.location = location

#========================================Object Creation Goes Below This Line============================================

#Wild Plains

WildPlains = Location("Wild Plains", "A large expanse of land covered with overgrowth; with blades of grass that reach your knees. You suspect there are at least one hundred species of insects living amongst the grass and there is a potent smell of dirt and plants.")

LostDog = NPC("Lost Dog", "A sweet little dog.")

LostDog.addline("Woof!")

Bandit = Enemy("Bandit","A man who will will try to steal your money.",15,4)

LostChange = Item("Lost Change", 3, "Three Gold Coins Left By An Unlucky Traveller", "Coins")

WildPlains.link\_npc(LostDog,"dog")

WildPlains.link\_npc(Bandit,"bandit")

WildPlains.link\_item(LostChange,"collection of coins")

#Flower Plains

FlowerPlains = Location("Flower Plains", " A fabulous area of green coated with wonderful flowers of all different colours. Owned by the city of kymar, this area of fields has been gardened to perfection. There is not a spot on the ground where you can’t see a beautiful flower has been grown.")

KymarGuard1 = NPC("Kymar Gaurd","Just a normal guard of Kymar City")

KymarGuard1.addline("Move Along")

GuardBadge = Item("Guard Badge", 10, "A Badge Used To Show The Employment As A Guard", "Badge")

KymarGuard1.additem(GuardBadge)

Gardener = NPC("Scruffy Gardener","A scruffy looking gardener who is busy tending to some flowers.")

Gardener.addline("Roses Look Wonderful In The Sunlight")

GardenRake = Item("Garden Rake", 2,"A Simple Rake Used To Tend The Flower Plains", "Tool")

Gardener.additem(GardenRake)

Tourist1 = NPC("Tourist", "A woman wearing foreign clothes admiring the surrounding flowers")

Tourist1.addline("I’ve never seen these many flowers before")

FlowerPlains.link\_npc(KymarGuard1,"guard")

FlowerPlains.link\_npc(Gardener,"gardener")

#Jonak City

LowerDistrict = Location("Jonak City | Lower District", "The poorest section of Jonak City, most manual labour is done by the citizens of this sector. The only part to have any money poured into it is the road that leads to the richer districts. There is a disgusting smell of animal waste as cattle do their business on the side of the road.")

Beggar = NPC("Beggar", "A very poor man begging for money since he seems to have none.")

Beggar.addline("Please, can you spare a coin?")

SpareChange = Item("Spare Change", 1, "A Coin Gifted To A Beggar", "Coin")

Beggar.additem(SpareChange)

PoorCitizen = NPC("Poor Citizen","A poor citizen tending to their farm, their only source of income.")

PoorCitizen.addline("Hopefully this year will yield a good harvest")

JonakGuard1 = NPC("Jonak Guard","Just a normal guard of Jonak City.")

JonakGuard1.addline("Move Along")

Thug = Enemy("Thug","Just A Mean Thug",5,5)

LowerDistrict.link\_npc(Beggar,"beggar")

LowerDistrict.link\_npc(PoorCitizen, "poor citizen")

LowerDistrict.link\_npc(JonakGuard1, "guard")

CentreDistrict = Location("Jonak City | Centre District", "The largest section of Jonak City, home to the middle class and the hub for almost all trade within the city. Although there is lots of money being made in this district, the majority of its inhabitants are the employees of these rich trade businesses.")

JonakGuard2 = NPC("Jonak Guard", "Just a normal guard of Jonak City")

JonakGuard2.addline("Move Along")

Citizen1 = NPC("Citizen","A normal citizen going to buy their weekly supplies.")

Citizen1.addline("I’ve got quite a busy day today")

Frank = Merchant("Frank", "A humble merchant willing to trade all for just a little coin.")

HonedBlade = Weapon("Honed Blade", 50, "A One-Handed Sword, Forged To Near Perfection And Made Out Of Steel.","Sword","subtraction",1,10)

ElvenWarhammer = Weapon("Elven Warhammer", 100, "A Large Hammer Made Out Of Elven Steel. Perfect For Cracking A Few Skeletons Apart.", "Hammer", "facts", 2, 20)

GrandHealthPotion = Potion("Grand Health Potion", 30, "A Large Health Potion That Can Be Used To Heal Someone", 50)

Frank.additem(HonedBlade)

Frank.additem(ElvenWarhammer)

Frank.additem(GrandHealthPotion)

CentreDistrict.link\_npc(JonakGuard2, "guard")

CentreDistrict.link\_npc(Citizen1, "citizen")

CentreDistrict.link\_npc(Frank, "merchant")

UpperDistrict = Location("Jonak City | Upper District", "The richest section of Jonak City. Home to Jonak’s Richest, this district if the most tantalising to look at. There are statues here, mansions there. A fancy water feature on every path and in every park. You notice that very little work in actually done in this district as it is likely made down in the lower sections.")

RichCitizen = NPC("Rich Citizen", "A rich citizen of Jonak, wearing much fancier clothes then most could ever dream of.")

RichCitizen.addline("Please stay away, I don’t want you dirtying my cloak.")

LargeGemstone = Item("Large Gemstone", 200, "A Fancy Gemstone, Probably Worth Quite A Bit To A Merchant", "Gemstone")

RichCitizen.additem(LargeGemstone)

Entertainer = NPC("Entertainer", "A centre district citizen trying to earn money by entertaining those with more money than him.")

EliteJonakGuard = NPC("Jonak Guard","A more weaponised version of the normal Jonak guard.")

EliteJonakGuard.addline("Move Along")

UpperDistrict.link\_npc(RichCitizen,"rich citizen")

UpperDistrict.link\_npc(Entertainer, "entertainer")

UpperDistrict.link\_npc(EliteJonakGuard, "elite guard")

#Dragon Claw Dungeon

Corridor = Location("Dragon Claw Dungeon | The Corridor", "A dark stone Corridor lit by the shine of the moon or sun outside. You see your shadow cast on the floor beneath as you walk through the entrance of Dragon Claw Dungeon.")

Spider = Enemy("Spider","A small spider, likely harmless",2,1)

Corridor.link\_npc(Spider,"spider")

WesternPassage = Location("Dragon Claw Dungeon | Western Passage", "A simple passageway to the west of the corridor, made from the same bleak stone as the rest of the dungeon.")

WesternSideRoom = Location("Dragon Claw Dungeon | Western Side-Room", "A small stone room, home to more bats and skeletons than actual treasure.")

Skeleton1 = Enemy("Skeleton","One Bony Boy",30,5,True)

Skeleton2 = Enemy("Skeleton","A Skeleton, With A Sword, And A Lust For Flesh",30,5,True)

WesternSideRoom.link\_npc(Skeleton1,"little skeleton")

WesternSideRoom.link\_npc(Skeleton2,"big skeleton")

EasternPassage = Location("Dragon Claw Dungeon | Eastern Passage","A passageway to the east of the corridor. There doesn’t seem to be anything important in this room. Unless... What’s that behind you?")

Ghost = Enemy("Ghost","A Spooky Ghost, Very Spooky",3,30,True)

EasternPassage.link\_npc(Ghost,"ghost")

EasternSideRoom = Location("Dragon Claw Dungeon | Eastern Side-Room", "A small room with a torch in it that seems to burn forever. It is one of the only rooms in the dungeon to have its own lighting and yet there is no warmth coming from the flame. There is some loot here though.")

StaffOfLightning = Weapon("Staff Of Lightning",50,"A Grand Staff With A Small Clear Ball Rested In The Top Of It. Inside Of The Ball You Can See Streaks Of Electricity Arch Across The Surfaces.","Staff","addition",2,15)

Emerald = Item("Emerald",45,"A Pretty Little Gem. Green In Colour, And Likely Worth Lots Of Gold.","Gemstone")

EasternSideRoom.link\_item(StaffOfLightning,"staff")

EasternSideRoom.link\_item(Emerald,"emerald")

GreatHall = Location("Dragon Claw Dungeon | The Great Hall", "A long hall made mostly out of oak wood. You reckon that this dungeon was previously someone’s house. Before, you know, they died and haunted the place forever.")

Bat = NPC("Bat", "A Sweet Little Bat, No Vampireness About This One")

Bat.addline("Squeak")

GreatHall.link\_npc(Bat,"bat")

BrokenBallroom = Location("Dragon Claw Dungeon | The Broken Ballroom", "A grand and luxurious ballroom, with a cracked marble floor and ripped curtains hanging over walls even though they lack windows. There is a sword rack on one of the walls.")

Zombie = Enemy("Zombie","Brains. Braaaaains. Braains.",20,20,True)

AncientSword = Weapon("Ancient Sword",30,"An Old Sword Left On A Sword Rack. You Cannot Be Sure What Race This Sword Was Made By. Perhaps A Race That Doesn’t Walk The Planet Anymore.","Sword","negatives",1,12)

BrokenBallroom.link\_npc(Zombie,"zombie")

BrokenBallroom.link\_item(AncientSword,"sword")

WindingStaircase = Location("Dragon Claw Dungeon | Winding Staircase", "A mesmerising spiral staircase that seems to infinitely descend deeper into the dungeon. Bats fly past your face as you follow the cracked stone steps and you swear you can hear a noise coming from the room below.")

VampireBat1 = Enemy("Vampire Bat","Bat + Vampire = Hungry For Blood",5,15,True)

WindingStaircase.link\_npc(VampireBat1, "vampire bat")

DeepCellar = Location("Dragon Claw Dungeon | Deep Cellar", "A dark cellar lined with rotten wood. Although previously used to store wine and food, all that is left is rats and… bones?")

DisgruntledSkeleton = Enemy("Disgruntled Skeleton", "A Rather Angry Looking Skeleton, If That’s Even Possible",10,4,True)

FastRat = Enemy("Fast Rat", "A small rat with fast legs and sharp teeth",5,20,True)

RottenFood = Item("Rotten Food",0,"A Pile Of Disgusting Rotten Food, Nibbled Away By Rats","Junk")

DeepCellar.link\_npc(DisgruntledSkeleton,"skeleton")

DeepCellar.link\_npc(FastRat,"rat")

DeepCellar.link\_item(RottenFood,"rotten food")

WesternTreasureRoom = Location("Dragon Claw Dungeon | Western Treasure Room", "A dimly lit room filled with treasure. Unless you have already looted the treasure, in which case the room is empty.")

BowOfAccurateShot = Weapon("Bow Of Accurate Shot",50,"A Keen Eye Is Still Needed To Use This Bow, But You May Find The Shots Land More Consistently Than Mundane Bows.","Bow","conversion",2,14)

WesternTreasureRoom.link\_item(BowOfAccurateShot,"bow")

EasternTreasureRoom = Location("Dragon Claw Dungeon | Eastern Treasure Room","A small stone room with multiple circles of soot on the walls. It seems as if these spots where exposed to some kind of flame.")

FireballSpell = Weapon("Fireball Spell Book", 65, "A Spell Tome That Can Be Read To Cast Fireball At Will.","Spell Book","floating",2,18)

EasternTreasureRoom.link\_item(FireballSpell,"spell book")

#The Deep Pit Dungeon

GreatFoyer = Location("The Deep Pit Dungeon | The Great Foyer","A large foyer with basalt walls. Strangely it gives the impression that this dungeon was designed to be just that, a dungeon.")

CrawlerZombie = Enemy("Crawler Zombie","A slow zombie, because this zombie has no legs.",10,20,True)

GreatFoyer.link\_npc(CrawlerZombie, "zombie")

ElongatedCorridor = Location("The Deep Pit Dungeon | Elongated Corridor","A long corridor that connects to many rooms. One of the rooms, an arena, is connected by a very large entrance made of human bones. A very powerful enemy rests in that room, it may be wise not to disturb it until you are strong enough to fight it.")

MassivePotion = Potion("Massive Health Potion",20,"An Extremely Large Health Potion, Guaranteed To Bring You To Full Health.",200)

ElongatedCorridor.link\_item(MassivePotion,"potion")

DarkSideRoom = Location("The Deep Pit Dungeon | Dark Side-Room","Other than the basalt walls that are shared with the rest of the dungeon. It seems that there is nothing special about this room. A waste of time really.")

RuinedTower = Location("The Deep Pit Dungeon | Ruined Tower", "A tower that used to stand great and tall. Unfortunately, the roof has since fallen in and there is no longer an elevation to the once great tower. Could be some treasure in the rubble though.")

VampireBat2 = Enemy("Count Batula", "The Famous Count Batula",25,32,True)

RuinedTower.link\_npc(VampireBat2,"vampire bat")

ElvenPassageway = Location("The Deep Pit Dungeon | Elven Passageway","The only room in this dungeon not to have basalt walls. This passageway contains uniquely elven architecture. Pillars with diorite shafts, golden capitals and brass plinths.")

Rats = Enemy("A Mischief Of Rats","There are a lot of rats, looks deadly",60,3)

ElvenPassageway.link\_npc(Rats,"bunch of rats")

DarkArena = Location("The Deep Pit Dungeon | The Dark Arena", "A sunken arena, home of the most fearsome creature housed in the deep pit dungeon, but also the rarest loot.")

LordHrungar = Enemy("Lord Hrungar","Only the final boss of the game, nothing massive.",150,20,True,True)

HrungarsBane = Weapon("Hrungar's Bane",666,"The Great-Sword Of The Mighty Lord Hrungar. Its Power Is More Than Any Mortal Could Imagine.","Greatsword","facts",2,50)

DarkArena.link\_npc(LordHrungar,"devil man")

DarkArena.link\_item(HrungarsBane,"greatsword")

SouthernLootRoom = Location("The Deep Pit Dungeon | Southern Loot Room","A small room filled with useless but likely valuable treasure. As you enter a sense of darkness washes over you.")

ShadowMonster = Enemy("Shadow Monster","A Monster made of shadows and darkness",15,28,True)

Diamond = Item("Diamond",80,"A Shiny Clear Gem. A Giant Covalent Structure Of Carbon Atoms You Believe.","Gemstone")

Topaz = Item("Topaz",30,"A Yellow Gem. Worth Lots To Any Merchant In The Trade.","Gemstone")

SouthernLootRoom.link\_npc(ShadowMonster,"shadow monster")

SouthernLootRoom.link\_item(Diamond,"diamond")

SouthernLootRoom.link\_item(Topaz,"topaz")

#Kymar city

NorthKymar = Location("North Kymar", "North Kymar is the residential area. Most citizens have houses here but there is very little trade or work.")

Citizen2 = NPC("Citizen","A citizen leaving their house. Probably on their way to work")

Citizen2.addline("Haven’t got the time to talk I’m afraid")

KymarGuard2 = NPC("Kymar Guard","He used to be an adventurer like you. Until he took an arrow to the knee.")

KymarGuard2.addline("Move Along")

NorthKymar.link\_npc(Citizen2,"citizen")

NorthKymar.link\_npc(KymarGuard2,"guard")

WestKymar = Location("West Kymar", "West Kymar connects directly to the flower plains, and, as such, is Kymar’s hub of trade and commerce.")

KymarGuard3 = NPC("Kymar Guard","Just a normal guard of Kymar City.")

KymarGuard3.addline("Move Along")

Tourist2 = NPC("Tourist","A tourist browsing a merchant’s products")

Tourist2.addline("They have so many items for sale here")

Notepad = Item("Notepad",2,"A basic notepad with leather cover and the pages held together with dried honeycomb.","Notepad")

Tourist2.additem(Notepad)

RichMerchant = Merchant("Rich Merchant","A merchant happy to sell you anything as long as you can afford it.")

OrcishDagger = Weapon("Orcish Dagger",140,"A Large Dagger Made In The Orcish Mountains. Not Designed For Precise Attacks.","Dagger","fixed",1,40)

DwarvenBattleaxe = Weapon("Dwarven Battle-Axe Of Flames",250,"An Enchanted Weapon Made By Ancient Dwarven Blacksmiths. Imbued With The Power Of Fire, This Battle-Axe Will Deal A Mighty Blow.","Battle-Axe","floating",2,50)

LesserPotion = Potion("Lesser Health Potion",10,"A Small Health Potion That Can Be Used To Heal Someone",10)

GreatPotion = Potion("Great Health Potion",25,"A Medium Sized Health Potion That Can Be Used To Heal Someone",30)

Trinket = Item("Shiny Trinket",100, "A Silver Trinket That Is Completely Useless. Worth A Good Penny Though.","Trinket")

RichMerchant.additem(OrcishDagger)

RichMerchant.additem(DwarvenBattleaxe)

RichMerchant.additem(LesserPotion)

RichMerchant.additem(GreatPotion)

WestKymar.link\_npc(KymarGuard3,"guard")

WestKymar.link\_npc(Tourist2,"tourist")

WestKymar.link\_npc(RichMerchant,"rich merchant")

SouthKymar = Location("South Kymar","South Kymar is farthest away from any visitors so is used as the centre of industry within the city. Lots of jobs are found here, as well as it being the source for lots of Kymar’s valuable exports.")

KymarGuard4 = NPC("Kymar Guard","Just a normal guard of Kymar City.")

KymarGuard4.addline("Move Along")

FactoryWorker = NPC("Factory Worker","A citizen currently working in the factory. Covered head to toe in grease, it looks like they haven’t showered in a while.")

FactoryWorker.addline("These products aren’t going to make themselves")

Wrench = Item("Wrench",5,"The Staple Tool Of Any Factory Worker","Tool")

FactoryWorker.additem(Wrench)

MachinePart = Item("Machine Part", 25, "A Part Of Some Sort Of Machine. Useless To You But Perhaps Worth Something To A Merchant.","Scrap")

SouthKymar.link\_npc(KymarGuard4,"guard")

SouthKymar.link\_npc(FactoryWorker,"factory worker")

SouthKymar.link\_item(MachinePart,"piece of scrap")

WildPlains.link\_location(LowerDistrict,"jonak city","the wild plains")

WildPlains.link\_location(Corridor,"the dragon's claw dungeon","the wild plains")

WildPlains.link\_location(FlowerPlains,"the flower plains","the wild plains")

LowerDistrict.link\_location(CentreDistrict,"the center district","the lower district")

CentreDistrict.link\_location(UpperDistrict,"the upper district","the center district")

FlowerPlains.link\_location(GreatFoyer,"the deep pit dungeon","the flower plains")

FlowerPlains.link\_location(WestKymar,"kymar city","the flower plains")

WestKymar.link\_location(NorthKymar,"north kymar","west kymar")

NorthKymar.link\_location(SouthKymar,"south kymar","north kymar")

SouthKymar.link\_location(WestKymar,"west kymar","south kymar")

Corridor.link\_location(WesternPassage,"the western passage","the corridor")

Corridor.link\_location(EasternPassage,"the eastern passage","the corridor")

WesternPassage.link\_location(WesternSideRoom,"the western side-room","the western passage")

EasternPassage.link\_location(EasternSideRoom,"the eastern side-room","the eastern passage")

WesternPassage.link\_location(GreatHall,"the great hall","the western passage")

GreatHall.link\_location(BrokenBallroom,"the broken ballroom","the great hall")

BrokenBallroom.link\_location(WindingStaircase,"the winding staircase","the broken ballroom")

WindingStaircase.link\_location(DeepCellar,"the deep cellar","the winding staircase")

DeepCellar.link\_location(WesternTreasureRoom,"the western treasure room","the deep cellar")

DeepCellar.link\_location(EasternTreasureRoom,"the eastern treasure room","the deep cellar")

GreatFoyer.link\_location(ElongatedCorridor,"the elongated corridor","the great foyer")

ElongatedCorridor.link\_location(DarkSideRoom,"the dark sideroom","the elongated corridor")

ElongatedCorridor.link\_location(RuinedTower,"the ruined tower","the elongated corridor")

ElongatedCorridor.link\_location(DarkArena,"the dark arena","the elongated corridor")

ElongatedCorridor.link\_location(ElvenPassageway,"the elven passage way","the elongated corridor")

ElvenPassageway.link\_location(SouthernLootRoom,"the southern loot room","the elven passageway")

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

player = Player() #Creates Player Object

player.move(LowerDistrict) #Puts Player In First Location

fists = Weapon("Bare Knuckles",0,"Your fists","Unarmed","conversion",2,4) #Creates The Starting Weapon "Fists"

player.equipedWeapon = fists #Sets Fists As The Equipped Weapon

def play(): #Creates Function So That Recursion Can Occur

if player.health <= 0: #Checks If The Player Is Dead

player.dead = True

command = input("> ")

commands = command.split(' ')

if commands[0].lower() == "goto":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument))

player.move(player.location.goto(commands[1]))

else:

type("Goto where?")

play() #Commands For Moving To A New Location

elif commands[0].lower() == "talkto":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument))

try:

player.location.connectedNPCs[commands[1]].speak()

except:

type("Wait who is the "+commands[1]+"?")

else:

type("Talk to who?")

play() #Command For Talking To A NPC

elif commands[0].lower() == "inv":

if len(commands) == 1:

type("=-----Player's Inventory -----=")

print()

for item in player.inventory:

player.inventory[item].invOutput()

print()

type("=-----------------------------=")

else:

type("Too Many Arguments For Inv, None Expected")

play() #Command For Outputting Inventory

elif commands[0].lower() == "stealfrom":

if len(commands) >1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument))

if commands[1].lower() in player.location.connectedNPCs:

print()

player.location.connectedNPCs[commands[1].lower()].outputInv()

print()

type("What would you like to steal?")

item = input("> ").title()

stolen = player.location.connectedNPCs[commands[1].lower()].steal(item)

if stolen is not None:

player.inventory[item] = stolen

type("You have stolen the "+stolen.get\_name())

else:

type("Wait who is the "+commands[1]+"?")

else:

type("Steal from who? ")

play() #Command For Stealing From A NPC

elif commands[0].lower() == "take":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

argument = " ".join(map(str,argument))

if argument in player.location.connectedItems:

item = player.location.connectedItems.pop(argument)

player.inventory[item.get\_name().title()] = item

type("You take the "+argument.title())

else:

type("There is no "+argument.title()+" here")

else:

type("Take what?") #Commands For Taking An Item From A Location

elif commands[0].lower() == "fight":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument))

if commands[1] in player.location.connectedNPCs:

enemy = player.location.connectedNPCs[commands[1]]

if isinstance(enemy,Enemy):

StartingHealth = enemy.health

while player.health > 0 and enemy.health > 0:

type("Your Health ["+str(player.health)+"]")

type("Their Health ["+str(enemy.health)+"]")

print()

type2("You swing to attack!")

hit = player.equipedWeapon.attack()

print()

enemy.health -= hit

type2("You dealt "+str(hit)+" damage!")

print()

if random.randint(0,1) == 1:

type2("The enemy attacks you!")

hit = enemy.attack()

player.health -= hit

type2("You took "+str(hit)+" damage!")

print()

if player.health <= 0:

player.dead = True

type2("You died, Game Over")

elif enemy.health <= 0:

type2("The enemy has been defeated!")

player.score += round((StartingHealth+enemy.damage)/2)

player.location.connectedNPCs.pop(commands[1])

type("Your score is now "+str(player.score))

if enemy.boss:

player.victorious = True

type("You have defeated the boss, you win!")

else:

type("You shouldn't kill civilians")

else:

type("Who is that?")

elif len(commands) > 2:

type("Too many arguments for Fight, Expected 1")

play()

else:

type("Fight who?")

play() #Command For Fighting A Non-Ambush Enemy

elif commands[0].lower() == "equiped":

type(player.equipedWeapon.get\_name()+" is the currently equiped weapon.") #Command For Outputting Currently Equipped Weapon

elif commands[0].lower() == "equip":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument))

if commands[1].title() in player.inventory:

player.equipedWeapon = player.inventory[commands[1].title()]

type("You have equipped "+player.equipedWeapon.get\_name())

else:

type("You do not have that weapon")

play()

else:

type("Equip what?")

play() #Command For Equipping A Weapon

elif commands[0].lower() == "sellto":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument)).title()

if commands[1].lower() in player.location.connectedNPCs:

npc = player.location.connectedNPCs[commands[1].lower()]

if isinstance(npc,Merchant):

type("=-----Player's Inventory -----=")

print()

for item in player.inventory:

player.inventory[item].invOutput()

print()

type("=-----------------------------=")

type("What item do you want to sell?")

item = input("> ")

if item.title() in player.inventory:

price = npc.sell(player.inventory.pop(item.title()))

player.cash += price

else:

type("You do not have that item!")

else:

type("You cannot sell to this person")

play()

else:

type("I don't see a "+commands[1].title())

else:

type("Sell to who?")

play() #Command For Selling To A Merchant

elif commands[0].lower() == "bal":

type("You have "+str(player.cash)+"₿") #Command For Outputing The Player's Balance

elif commands[0].lower() == "buyfrom":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

commands[1] = " ".join(map(str,argument)).title()

if commands[1].lower() in player.location.connectedNPCs:

npc = player.location.connectedNPCs[commands[1].lower()]

if isinstance(npc,Merchant):

npc.outputInv()

type("[Balance]: "+str(player.cash)+"₿")

type("What would you like to buy? ")

item = input("> ").title()

if item in npc.inventory:

if npc.inventory[item].get\_value() <= player.cash:

player.cash -= npc.inventory[item].get\_value()

player.inventory[item] = npc.inventory.pop(item)

type("You have bought the "+player.inventory[item].get\_name())

else:

type("You cannot afford that item!")

else:

type("That item is not for sale")

else:

type("You cannot buy from this person")

play()

else:

type("I don't see a "+commands[1].title())

else:

type("Buy from who?")

play() #Command For Buying Items From A Merchant

elif commands[0].lower() == "drink":

if len(commands) > 1:

argument = []

for word in commands:

if word != commands[0]:

argument.append(word)

argument = " ".join(map(str,argument)).title()

if argument in player.inventory:

item = player.inventory.pop(argument)

if isinstance(item,Potion):

regen = item.use()

type("You heal "+str(regen)+" health")

player.health += regen

type("You drink the "+argument.title())

if player.health > 100:

player.health = 100

else:

type("You cannot drink that")

else:

type("You don't have a "+argument.title())

else:

type("Drink what?") #Command For Drinking A Potion

elif commands[0].lower() == "health":

type("You have "+str(player.health)+" health") #Command For Outputing Current Health

elif commands[0].lower() == "help":

type("Commands:")

type("goto : move to a location")

type("talkto : talk to an npc")

type("inv : view your inventory")

type("stealfrom : steal from an npc")

type("fight : fight with an enemy npc")

type("help : view this command list")

type("equiped : shows you what weapon is currently equiped")

type("equip : equip a weapon from your inventory")

type("sellto : sell to a merchant npc")

type("bal : view your current balance")

type("buyfrom : buy items from a merchant")

type("drink : drink a potion to restore health")

type("health : view your current health") #Command For Outputting What Each Command Does

else:

type("I Don't Know How To Do That!")

play()

print()

type("Use help to view commands, By default answers are 8 bit")

print()

while not player.dead and not player.victorious: #Main Loop That Stops When The Player Is Dead Or Has Won

player.location.info()

defeated\_ambushes = []

for npc in player.location.connectedNPCs:

if isinstance(player.location.connectedNPCs[npc],Enemy):

if player.location.connectedNPCs[npc].ambush:

type("You are attacked by "+player.location.connectedNPCs[npc].get\_name()+" the "+npc) #Starts Fight If Ambushing Enemy Is In Location

print()

enemy = player.location.connectedNPCs[npc]

while player.health > 0 and enemy.health > 0:

type("Your Health ["+str(player.health)+"]")

type("Their Health ["+str(enemy.health)+"]")

print()

type2("You swing to attack!")

hit = player.equipedWeapon.attack()

print()

enemy.health -= hit

type2("You dealt "+str(hit)+" damage!")

print()

if random.randint(0,1) == 1:

type2("The enemy attacks you!")

hit = enemy.attack()

player.health -= hit

type2("You took "+str(hit)+" damage!")

print()

if player.health <= 0:

player.dead = True

type2("You died, Game Over")

elif enemy.health <= 0:

type2("The enemy has been defeated!")

player.score += round((enemy.health+enemy.damage)/2)

defeated\_ambushes.append(npc)

type("Your score is now "+str(player.score))

for npc in defeated\_ambushes:

player.location.connectedNPCs.pop(npc)

play() #Calls The Ability To Type Commands

name = username

try:

file = open('scoreboard.csv','x') #Will create Scoreboard.csv if it doesn't already exist

file.close()

except:

pass

finally:

scoreboard = []

file = open('scoreboard.csv','r') #Reads The Scoreboard, Adds The Player's Score, Sorts It, And Outputs The Scores

for line in file:

line = line.split(',')

line[1] = line[1].rstrip()

scoreboard.append(line)

sorted = []

scoreboard.append([name,str(player.score)])

while len(scoreboard) > 0: #Sorts The scoreboard By The Score Values

for item in scoreboard:

count = 0

for item2 in scoreboard:

if int(item[1]) >= int(item2[1]):

count += 1

if count == (len(scoreboard)):

scoreboard.remove(item)

sorted.append(item)

for score in sorted:

type(score[0]+": "+score[1])

file.close()

file = open('scoreboard.csv','w') #ReWrites The Scoreboard With The New Scores

for score in sorted:

file.write(score[0]+","+score[1]+"\n")

file.close()

Items.py

import sys, time, random

from textadventure.questions import Question

def type(string):

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.02)

sys.stdout.write("\n")

sys.stdout.flush()

class Item:

"""Parent Class For All Items"""

def \_\_init\_\_(self,name,value,description,type):

self.name = name #Name Of Item

self.value = value #How Much An Item Is Worth

self.description = description #Description Of Item

self.type = type #What Type Of Item It Is

def get\_name(self):

"""Returns Name Attribute Of Item Object"""

return self.name

def get\_value(self):

"""Returns Value Attribute Of Item Object"""

return self.value

def get\_description(self):

"""Returns Description Attribute Of Item Object"""

return self.description

def get\_type(self):

"""Returns Type Attribute Of Item Object"""

return self.type

def set\_name(self,name):

"""Sets The Name Of The Item Object"""

self.name = name

def set\_value(self,value):

"""Sets The Value Of The Item Object"""

self.value = value

def set\_description(self,description):

"""Sets The Description Of The Item Object"""

self.description = description

def set\_type(self,type):

"""Sets The Type Of The Item Object"""

self.type = type

def itemInfo(self,place):

"""Outputs Information About Itself"""

type("There is a"+" "+self.type+" "+"in this"+" "+place)

type("=-----------------------=")

type(self.description)

print()

def invOutput(self):

"""Outputs Inventory Version Of Information About Itself"""

type("Name: "+self.name)

type("Description: "+self.description)

type("Value: "+str(self.value))

type("Type: "+self.type)

class Weapon(Item):

"""Item Sub-class Used For Fighting"""

def \_\_init\_\_(self,name,value,description,type,damageType,hands,damage):

super().\_\_init\_\_(name,value,description,"Weapon") #Calls Parent \_\_init\_\_

self.damageType = damageType #The Type Of Damage The Weapon Deals

self.weaponType = type #The Type Of Weapon It Is

self.hands = hands #The Number Of Hands Needed To Use The Weapon

self.damage = damage #The Maximum Damage The Weapon Can Deal

def get\_damageType(self):

"""Returns Damage Type Attribute Of Weapon Object"""

return self.damageType

def get\_WeaponType(self):

"""Returns Weapon Type Attribute Of Weapon Object"""

return self.WeaponType

def get\_hands(self):

"""Returns Number Of Hands Attribute Of Weapon Object"""

return self.hands

def set\_damageType(self,damageType):

"""Sets Damage Type Attribute Of Weapon Object"""

self.damageType = damageType

def set\_weaponType(self,weaponType):

"""Sets Weapon Type Attribute Of Weapon Object"""

self.weaponType = weaponType

def set\_hands(self,hands):

"""Returns Number Of Hands Attribute Of Weapon Object"""

self.hands = hands

def attack(self):

"""Generates Questions Based On Damage Type, And Returns Damage Value Depending On If The Question Was Answered Correctly"""

if self.damageType.lower() == "addition":

if Question().addition():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType.lower() == "subtraction":

if Question().subtraction():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType == "negatives":

if random.randint(0,1) == 1:

if Question().negToBin():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

else:

if Question().binToNeg():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType.lower() == "fixed":

if random.randint(0,1) == 1:

if Question().fixToBin():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

else:

if Question().binToFixed():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType.lower() == "floating":

if random.randint(0,1) == 1:

if Question().floatToBin():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

else:

if Question().binToFloat():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType.lower() == "facts":

if Question().multipleChoice():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif self.damageType.lower() == "conversion":

num = random.randint(0,5)

if num == 0:

if Question().decToBinConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif num == 1:

if Question().binToDecConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif num == 2:

if Question().decToHexConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif num == 3:

if Question().hexToDecConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif num == 4:

if Question().binToHexConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

elif num == 5:

if Question().hexToBinConv():

return random.randint(round(self.damage/2),self.damage)

else:

return random.randint(0,round(self.damage/10))

def invOutput(self):

"""Override For Item InvOutput To Output Weapon Attributes Aswell"""

type("Name: "+self.name)

type("Description: "+self.description)

type("Value: "+str(self.value))

type("Type: "+self.weaponType)

type("Damage Type: "+self.damageType.title())

type("Number Of Hands: "+str(self.hands))

type("Max Damage: "+str(random.randint(round(self.damage/2),self.damage)))

class Potion(Item):

"""Item Sub-class Used For Health Potions"""

def \_\_init\_\_(self,name,value,description,health):

super().\_\_init\_\_(name,value,description,"Health Potion") #Calls Parent Init, Sets Type To Health Potion

self.health = health #The Maximum Health The Potion Can Heal

def use(self):

"""Returns A Half-Random Health Value Based On The Health Attribute"""

return random.randint(round(self.health/2),self.health)

Npcs.py

import sys,time,random

def type(string):

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.02)

sys.stdout.write("\n")

sys.stdout.flush()

class NPC:

"""Parent Class For All Non-Player Characters"""

def \_\_init\_\_(self,name,description):

self.name = name #The Name Of The NPC

self.description = description #A Description Of The NPC

self.conversation = [] #The Lines Of Conversation The NPC Can Say

self.inventory = {} #A Dictionary To Contain What Items The NPC Has

def get\_name(self):

"""Returns Name Attribute Of NPC Object"""

return self.name

def get\_description(self):

"""Returns Description Attribute Of NPC Object"""

return self.description

def set\_name(self,name):

"""Sets Name Attribute Of NPC Object"""

self.name = name

def set\_description(self,description):

"""Sets Description Attribute Of NPC Object"""

self.description = description

def addline(self,line):

"""Adds A Conversation Line To The Array"""

self.conversation.append(line)

def additem(self,item):

"""Adds An Item Object To The Inventory Dictionary"""

self.inventory[item.get\_name()] = item

def outputInv(self):

"""Outputs The NPC's Inventory"""

type("=-----"+self.name+"'s Inventory -----=")

for item in self.inventory:

self.inventory[item].invOutput()

print()

line = ["-" for char in range(len("=-----"+self.name+"'s Inventory -----=")-2)]

type("="+"".join(map(str,line))+"=")

def steal(self,item):

"""Returns And Pops A Stolen Item Object"""

if item in self.inventory:

return self.inventory.pop(item)

else:

type("They do not have that item")

return None

def speak(self):

"""Outputs The Conversation Array"""

for line in self.conversation:

type("["+self.name+"]: "+line)

class Enemy(NPC):

"""NPC Sub-class Used For People That Fight The Player"""

def \_\_init\_\_(self,name,description,health,damage,ambush=False,boss=False):

super().\_\_init\_\_(name,description) #Calls Parent \_\_init\_\_

self.health = health #The Health Of The Enemy

self.damage = damage #The Maximum Damage The Enemy Can Deal

self.ambush = ambush #Whether Or Not The Enemy Will Attack The Player By Themselves

self.boss = boss #Wether Or Not This Enemy Is The Final Boss

def set\_health(self,health):

"""Sets Health Attribute Of Enemy Object"""

self.health = health

def set\_damage(self, damage):

"""Sets Damage Attribute Of Enemy Object"""

self.damage = damage

def get\_health(self):

"""Returns Health Attribute Of Enemy Object"""

return self.health

def get\_damage(self):

"""Returns Damage Attribute Of Enemy Object"""

return self.damage

def hurt(self,damage):

"""Reduces The Enemy's Health By A Given Amount"""

self.health -= damage

def attack(self):

"""Returns A Half-Random Damage Value Based On The Damage Attribute"""

return random.randint(round(self.damage/2),self.damage)

class Merchant(NPC):

"""NPC Sub-class Used For People Who Can Be Traded With"""

def \_\_init\_\_(self,name,description):

super().\_\_init\_\_(name,description) #Calls Parent \_\_init\_\_

self.name += " the merchant" #Changes Name To Contain "The Merchant"

def sell(self,item):

"""Outputs A Confirmation Of An Item Being Sold, And Returns Item Value"""

type("You sell your "+item.get\_name()+" to "+self.name)

print()

return item.get\_value()

def steal(self,item):

"""Overrides The NPC Steal Method So That Merchants Cannot Be Stolen From"""

type("You cannot steal from merchants")

return None

Gps.py

import sys,time

def type(string):

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.015)

sys.stdout.write("\n")

sys.stdout.flush()

class Location:

"""Main Location Object"""

def \_\_init\_\_(self,name,description):

self.name = name #Name Of Location

self.description = description #Description Of Location

self.connectedLocations = {} #Dictionary Of Connected Locations

self.connectedNPCs = {} #Dictionary Of Connected NPCs

self.connectedItems = {} #Dictionary Of Connected Items

def get\_name(self):

"""Returns Name For Location Object"""

return self.name

def get\_description(self):

"""Returns Description For Location Object"""

return self.description

def get\_connectedLocations(self):

"""Returns The Dictionary Of Connected Locations For The Location Object"""

return self.connectedLocations

def get\_connectedNPCs(self):

"""Returns The Dictionary Of Connected NPCs For The Location Object"""

return self.connectedNPCs

def get\_connectedItems(self):

"""Returns The Dictionary Of Connected Items For The Location Object"""

return self.connectedItems

def set\_name(self,name):

"""Sets The Name Attribute"""

self.name = name

def set\_description(self,description):

"""Sets The Description Attribute"""

self.description = description

def link\_location(self,location,type,type2):

"""Adds A Location To The Dictionary Of Connected Locations"""

self.connectedLocations[type] = location

location.connectedLocations[type2] = self

def link\_npc(self,npc,ShortDescription):

"""Adds A NPC To The Dictionary Of Connected NPCs"""

self.connectedNPCs[ShortDescription] = npc

def link\_item(self,item,type):

"""Adds An Item To The Dictionary Of Connected Items"""

self.connectedItems[type] = item

def goto(self,place):

"""Returns A Valid Location From The Connected Locations Dictionary Based On Input"""

if place.lower() in self.connectedLocations:

return self.connectedLocations[place.lower()]

else:

type("There is no"+" "+place+" "+"here")

return self

def info(self):

"""Outputs Infomartion About Itself"""

type(self.name)

line = ["-" for char in range(len(self.name)-2)]

type("="+"".join(map(str,line))+"=")

type(self.description.title())

line = ["-" for char in range(118)]

type("="+"".join(map(str,line))+"=")

for place in self.connectedLocations:

type("There is an enterance to "+place.title())

for npc in self.connectedNPCs:

type("There is a "+npc.title()+" here!")

for item in self.connectedItems:

type("You can see a "+item.title())

print()

Questions.py

import random, sys, time

def type(string):

for char in string:

sys.stdout.write(char)

sys.stdout.flush()

time.sleep(0.02)

sys.stdout.write("\n")

sys.stdout.flush()

class Question:

"""Class For Housing Question Generation"""

def \_\_init\_\_(self):

pass #Has No Attributes

def addition(self):

"""Generates An Addition Question"""

num1 = random.randint(0,255)

num2 = random.randint(0,255)

num = num1

binary1 = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary1.append(1)

else:

binary1.append(0)

num = num2

binary2 = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num-= power

binary2.append(1)

else:

binary2.append(0)

type("Add the binary numbers "+"".join(map(str,binary1))+" and "+"".join(map(str,binary2))+". Input your answer as a 9 bit number")

answer = input(">>> ")

numOut = num1 + num2

num = numOut

binaryOut = []

for x in range(9):

power = 2\*\*(8-x)

if num - power >= 0:

num-= power

binaryOut.append(1)

else:

binaryOut.append(0)

answerList = []

for char in answer:

try:

answerList.append(int(char))

except:

pass

if len(answerList) == 8:

answerList.insert(0, 0)

if answerList == binaryOut:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+"".join(map(str,binaryOut)))

return False

def decToBinConv(self):

"""Generates A Decimal To Binary Conversion Question"""

dec = random.randint(0,255)

type("Conver "+str(dec)+" to binary")

num = dec

binary = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary.append(1)

else:

binary.append(0)

binaryStr = "".join(map(str,binary))

answer = input(">>> ")

if binaryStr == answer:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+binaryStr)

return False

def binToDecConv(self):

"""Generates A Binary To Decimal Conversion Question"""

dec = random.randint(0,255)

num = dec

binary = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary.append(1)

else:

binary.append(0)

binaryStr = "".join(map(str,binary))

type("Convert "+binaryStr+" to decimal")

answer = input(">>> ")

try:

intAnswer = int(answer)

except:

intAnswer = (-1)

if dec == intAnswer:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+str(dec))

return False

def decToHexConv(self):

"""Generates A Decimal To Hexadecimal Conversion Question"""

dec = random.randint(0,255)

first = dec//16

second = dec - (16\*first)

if first == (10):

first = "A"

elif first == (11):

first = "B"

elif first == (12):

first = "C"

elif first == (13):

first = "D"

elif first == (14):

first ="E"

elif first == (15):

first = "F"

if second == (10):

second = "A"

elif second == (11):

second = "B"

elif second == (12):

second = "C"

elif second == (13):

second = "D"

elif second == (14):

second ="E"

elif second == (15):

second = "F"

type("Convert "+str(dec)+" to hexadecimal")

answer = input(">>> ")

hex = str(first)+str(second)

if answer.upper() == hex:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+hex)

return False

def binToHexConv(self):

"""Generates A Binary To Hexadecimal Conversion Question"""

dec = random.randint(0,255)

num = dec

binary = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary.append(1)

else:

binary.append(0)

binaryStr = "".join(map(str,binary))

type("Convert "+binaryStr+" to hexadecimal")

answer = input(">>> ")

first = dec//16

second = dec - (16\*first)

if first == (10):

first = "A"

elif first == (11):

first = "B"

elif first == (12):

first = "C"

elif first == (13):

first = "D"

elif first == (14):

first ="E"

elif first == (15):

first = "F"

if second == (10):

second = "A"

elif second == (11):

second = "B"

elif second == (12):

second = "C"

elif second == (13):

second = "D"

elif second == (14):

second ="E"

elif second == (15):

second = "F"

hex = str(first)+str(second)

if answer.upper() == hex:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+hex)

return False

def hexToBinConv(self):

"""Generates A Hexadecimal To Binary Conversion Question"""

dec = random.randint(0,255)

first = dec//16

second = dec - (16\*first)

if first == (10):

first = "A"

elif first == (11):

first = "B"

elif first == (12):

first = "C"

elif first == (13):

first = "D"

elif first == (14):

first ="E"

elif first == (15):

first = "F"

if second == (10):

second = "A"

elif second == (11):

second = "B"

elif second == (12):

second = "C"

elif second == (13):

second = "D"

elif second == (14):

second ="E"

elif second == (15):

second = "F"

hex = str(first)+str(second)

type("Convert "+hex+" (Hex) to binary")

num = dec

binary = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary.append(1)

else:

binary.append(0)

binaryStr = "".join(map(str,binary))

answer = input(">>> ")

if answer == binaryStr:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+binaryStr)

return False

def hexToDecConv(self):

"""Generates A Hexadecimal To Decimal Conversion Question"""

dec = random.randint(0,255)

first = dec//16

second = dec - (16\*first)

if first == (10):

first = "A"

elif first == (11):

first = "B"

elif first == (12):

first = "C"

elif first == (13):

first = "D"

elif first == (14):

first ="E"

elif first == (15):

first = "F"

if second == (10):

second = "A"

elif second == (11):

second = "B"

elif second == (12):

second = "C"

elif second == (13):

second = "D"

elif second == (14):

second ="E"

elif second == (15):

second = "F"

hex = str(first)+str(second)

type("Convert "+hex+" (Hex) to decimal")

answer = input(">>> ")

if answer == str(dec):

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+str(dec))

return False

def subtraction(self):

"""Generates A Binary Subtraction Question"""

num1 = random.randint(0,255)

num2 = random.randint(0,255)

num = num1

binary1 = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binary1.append(1)

else:

binary1.append(0)

num = num2

binary2 = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num-= power

binary2.append(1)

else:

binary2.append(0)

binaryStr1 = "".join(map(str,binary1))

binaryStr2 = "".join(map(str,binary2))

type("Subtract "+binaryStr1+" from ",binaryStr2)

calc = num2 - num1

num = calc

binaryOut = []

for x in range (9):

power = 2\*\*(8-x)

if x == 0:

power = -power

if num - power >= 0:

num -= power

binaryOut.append(1)

else:

binaryOut.append(0)

answer = input(">>> ")

binaryStrOut = "".join(map(str,binaryOut))

if answer == binaryStrOut:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+binaryStrOut)

def negToBin(self):

"""Generates A Negative Decimal To Signed Binary Conversion Question"""

num = random.randint(-256,0)

type("Convert "+str(num)+" to a signed binary number")

binaryOut = []

for x in range (9):

power = 2\*\*(8-x)

if x == 0:

power = -power

if num - power >= 0:

num -= power

binaryOut.append(1)

else:

binaryOut.append(0)

answer = input(">>> ")

binaryStr = "".join(map(str,binaryOut))

if answer == binaryStr:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+binaryStr)

return False

def binToNeg(self):

"""Generates A Signed Binary To Negative Decimal Conversion Question"""

number = random.randint(-256,0)

num = number

binaryOut = []

for x in range (9):

power = 2\*\*(8-x)

if x == 0:

power = -power

if num - power >= 0:

num -= power

binaryOut.append(1)

else:

binaryOut.append(0)

binaryStr = "".join(map(str,binaryOut))

type("Convert "+binaryStr+" to a negative decimal")

answer = input(">>> ")

if answer == str(number):

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+str(number))

return False

def fixToBin(self):

"""Generates A Fixed Point Decimal To Binary Conversion Question"""

number = round(random.uniform(0, 15.9375),4)

numberStr = str(number)

numberList = numberStr.split(".")

integers = int(numberList[0])

decimals = float("0."+numberList[1])

num = integers

binaryInt = []

for x in range(4):

power = 2\*\*(3-x)

if num - power >= 0:

num -= power

binaryInt.append(1)

else:

binaryInt.append(0)

binaryIntStr = "".join(map(str,binaryInt))

num = (decimals)

binaryDec = []

for x in range(4):

power = (1/(2\*\*(x+1)))

if num - power >= 0:

num -= power

binaryDec.append(1)

else:

binaryDec.append(0)

binaryDecStr = "".join(map(str,binaryDec))

binary = binaryIntStr+binaryDecStr

type("Convert "+str(number)+" to a fixed point, 8 bit, binary number (4bits.4bits)")

answer = input(">>> ")

if answer == binary:

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+binary)

return False

def binToFixed(self):

"""Generates A Binary To Fixed Point Decimal Conversion Question"""

number = 0.625\*round(random.uniform(0, 15.9375)/0.0625)

print(number)

numberStr = str(number)

numberList = numberStr.split(".")

integers = int(numberList[0])

decimals = float("0."+numberList[1])

num = integers

binaryInt = []

for x in range(4):

power = 2\*\*(3-x)

if num - power >= 0:

num -= power

binaryInt.append(1)

else:

binaryInt.append(0)

binaryIntStr = "".join(map(str,binaryInt))

num = (decimals)

binaryDec = []

for x in range(4):

power = (1/(2\*\*(x+1)))

if num - power >= 0:

num -= power

binaryDec.append(1)

else:

binaryDec.append(0)

binaryDecStr = "".join(map(str,binaryDec))

binary = binaryIntStr+binaryDecStr

type("Convert the fixed point binary number: "+binary+"to a decimal number")

answer = input(">>> ")

if answer == str(number):

type("Correct")

return True

else:

type("Incorrect")

type("The answer was "+str(number))

return False

def floatToBin(self):

"""Generates A Floating Point Decimal To Binary Conversion Question"""

number = 0.625\*round(random.uniform(0,128)/0.625)

if number < 1:

number = 0.625\*round(number/0.0625)

elif number < 10:

number = 0.125\*round(number/0.125)

elif number < 100:

number = 0.25\*round(number/0.25)

else:

number = 0.5\*round(number/0.5)

numberStr = str(number)

numberList = numberStr.split(".")

integers = int(numberList[0])

decimals = float("0."+numberList[1])

num = integers

binaryInt = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binaryInt.append(1)

elif (1) in binaryInt:

binaryInt.append(0)

binaryIntStr = "".join(map(str,binaryInt))

num = (decimals)

binaryDec = []

for x in range(8-len(binaryInt)):

power = (1/(2\*\*(x+1)))

if num - power >= 0:

num -= power

binaryDec.append(1)

else:

binaryDec.append(0)

binaryDecStr = "".join(map(str,binaryDec))

exponent = (len(binaryInt)-1)

num = exponent

binaryExp = []

for x in range(3):

power = (2\*\*(2-x))

if num - power >= 0:

num -= power

binaryExp.append(1)

else:

binaryExp.append(0)

binaryExpStr = "".join(map(str,binaryExp))

solution = binaryIntStr+binaryDecStr+" "+binaryExpStr

type("Convert the number "+str(number)+" to an 8 bit floating point binary number, with a 3 bit exponent. E.g. 45 = 10110100 101")

answer = input(">>> ")

if answer == solution:

type("Correct")

return True

else:

type("Incorrect")

type("The Answer was "+solution)

return False

def binToFloat(self):

"""Generates A Binary To Floating Point Decimal Conversion Question"""

number = 0.625\*round(random.uniform(0,128)/0.625)

if number < 1:

number = 0.625\*round(number/0.0625)

elif number < 10:

number = 0.125\*round(number/0.125)

elif number < 100:

number = 0.25\*round(number/0.25)

else:

number = 0.5\*round(number/0.5)

numberStr = str(number)

numberList = numberStr.split(".")

integers = int(numberList[0])

decimals = float("0."+numberList[1])

num = integers

binaryInt = []

for x in range(8):

power = 2\*\*(7-x)

if num - power >= 0:

num -= power

binaryInt.append(1)

elif (1) in binaryInt:

binaryInt.append(0)

binaryIntStr = "".join(map(str,binaryInt))

num = (decimals)

binaryDec = []

for x in range(8-len(binaryInt)):

power = (1/(2\*\*(x+1)))

if num - power >= 0:

num -= power

binaryDec.append(1)

else:

binaryDec.append(0)

binaryDecStr = "".join(map(str,binaryDec))

exponent = (len(binaryInt)-1)

num = exponent

binaryExp = []

for x in range(3):

power = (2\*\*(2-x))

if num - power >= 0:

num -= power

binaryExp.append(1)

else:

binaryExp.append(0)

binaryExpStr = "".join(map(str,binaryExp))

solution = binaryIntStr+binaryDecStr+" "+binaryExpStr

type("Convert the number "+solution+" decimal number E.g. 10110100 101 = 45 ")

answer = input(">>> ")

if int(answer) == (number):

type("Correct")

return True

else:

type("Incorrect")

type("The Answer was "+str(number))

return False

def multipleChoice(self):

"""Generates A Multiple Choice Question"""

correct = [

"ASCII Uses 7 Bits",

"Unicode Uses Up To 48 Bits",

"Parity Checks Are Used To Check That Recieved Data Is Correct",

"In Majority Voting The Modal Bit Recieved Is Used",

"Checksums Use Unique Algorithms To Check If Data Recieved Is Correct",

"Checkdigits Are Specific Checksums That Product Only One Extra Digit",

"Lossy Compression Loses Data When Compressing Files",

"Lossless Compression Doesn't Lose Data When Compressing Files",

"Run length encoding removes repeated information and replaces it with one occurrence of the repeated information aswell as the number of times it is repeated",

"When a file is compressed with a dictionary-based method, a dictionary containing repeated data is appended to the file",

"Encryption is the process of scrambling data so that it cannot be understood if intercepted",

"Caesar ciphers encrypt information by replacing characters",

"The Vernam cipher is an example of a one-time pad cipher. "

]

incorrect = [

"ASCII Uses 6 Bits",

"Unicode Uses Up To 64 Bits",

"Parity Checks Are Used To Check That Data Was Recieved",

"In Majority Voting The Least Modal Bit Recieved Is Used",

"Checksums Use Unique Algorithms To Check If Data Was Recieved",

"Checkdigits Are Specific Checksums That Product Only Six Extra Digits",

"Lossy Compression Doesn't Lose Data When Compressing Files",

"Lossless Compression Loses Data When Compressing Files",

"Run length encoding increases the size of a file",

"When a file is compressed with a dictionary-based method, a dictionary containing repeated data is removed from the file",

"Encryption is the process of unscrambling data so that it can be understood if intercepted",

"Caesar ciphers are a type of food made by Caesar",

"The Vernam cipher is a type of food made by Verman",

"Hexadecimal is a fancy shape invented by a mathematician",

"Binary is a base 4 number system",

"Hexadecimal is a base 32 number system",

"ASCII uses 12 bits",

"When adding 1 and 1 the result is 1",

"Compression makes files larger",

"When the Modal bit recieved is used that is called Minority voting",

"11 In Hexadecimal is F",

"Unicode uses 128 bits",

"Run length encoding is a form of Encryption",

"The caesar cipher is an example of one-time pad cipher",

"The Vernam cipher encrypts information by replacing characters",

"There are no errors when data is transmitted from computer to computer",

"Odd Parity works to make the number of 1s even",

"Even Parity works to make the number of 1s odd",

"When Majority voting is being used each bit is only transmitted once",

"Checksums can correct errors",

"With lossy compression file size can be reduced without impacting the quality of the file",

"An XOR is not used in a Vernam Cipher",

"No extra data needs to be added with dictionary-based compression",

"Encrypted information is known as plaintext",

"Unencrypted information is known as ciphertext",

"The Vernam cipher uses a non-random key"

]

letters = ["A","B","C","D"]

correctLetter = random.choice(letters)

letters.remove(correctLetter)

output = [None,None,None,None]

if correctLetter == "A":

output[0] = "[A]"+" "+random.choice(correct)

elif correctLetter == "B":

output[1] = "[B]"+" "+random.choice(correct)

elif correctLetter == "C":

output[2] = "[C]"+" "+random.choice(correct)

elif correctLetter == "D":

output[3] = "[D]"+" "+random.choice(correct)

for item in range(len(output)):

if output[item] is None:

incorrectans = random.choice(incorrect)

if item == 0:

output[item] = "[A]"+" "+incorrectans

elif item == 1:

output[item] = "[B]"+" "+incorrectans

elif item == 2:

output[item] = "[C]"+" "+incorrectans

elif item == 3:

output[item] = "[D]"+" "+incorrectans

incorrect.remove(incorrectans)

type("Input the letter of the true answer")

for item in output:

print(item)

answer = input(">>> ")

if answer.upper() == correctLetter:

type("Correct")

return True

else:

type("Incorrect")

type("The correct answer was "+correctLetter)

return False

ˍˍinitˍˍ.py

from .gps import Location

from .items import Item, Weapon, Potion

from .npcs import NPC, Enemy, Merchant

from .questions import Question