Lucas Armyn

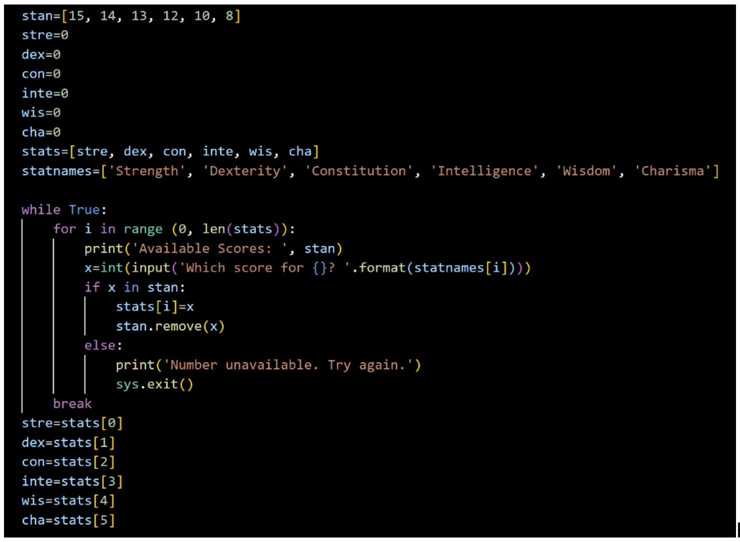
ENAE380 Final Project

Design PDF

1/20/23

Section 1: Introduction

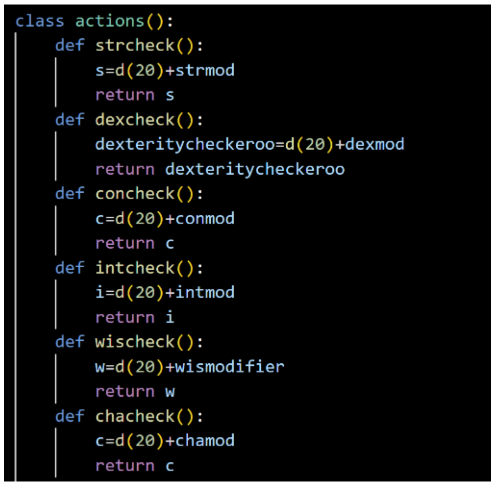
The project that I chose to create is a text-based adventure game based on the tabletop role playing game Dungeons and Dragons. The game uses a simple random number generator with accompanying bonuses and penalties to create ‘skill checks’, a dice roll whenever the player attempts a task which determines success or failure. At the beginning of the game, the player determines their statistics, Strength, Dexterity, Constitution, Intelligence, Wisdom, and Charisma, shown in figure 1 below. They then allocate values of 15, 14, 13, 12, 10, and 8 to each of these statistics, granting them an accompanied numerical bonus which is added onto the randomly generated die roll number between 1 and 20, simulating rolling a 20-sided die, the stat-specific dice roll functions shown in figure 2. The function for the dice roll is shown below in figure 3.



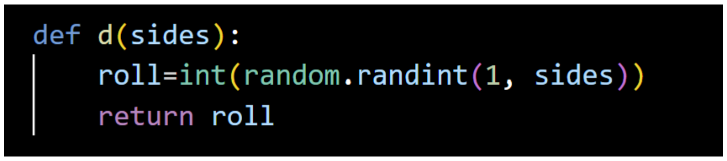
```python
stan=[15, 14, 13, 12, 10, 8]
stre=0
dex=0
con=0
inte=0
wis=0
cha=0
stats=[stre, dex, con, inte, wis, cha]
statnames=['Strength', 'Dexterity', 'Constitution', 'Intelligence', 'Wisdom', 'Charisma']

while True:
    for i in range (0, len(stats)):
        print('Available Scores: ', stan)
        x=int(input('Which score for {}? '.format(statnames[i])))
        if x in stan:
            stats[i]=x
            stan.remove(x)
        else:
            print('Number unavailable. Try again.')
            sys.exit()
    break
stre=stats[0]
dex=stats[1]
con=stats[2]
inte=stats[3]
wis=stats[4]
cha=stats[5]
```

Fig. 1



```
class actions():
    def strcheck():
        s=d(20)+strmod
        return s
    def dexcheck():
        dexteritycheckeroo=d(20)+dexmod
        return dexteritycheckeroo
    def concheck():
        c=d(20)+conmod
        return c
    def intcheck():
        i=d(20)+intmod
        return i
    def wischeck():
        w=d(20)+wismodifier
        return w
    def chacheck():
        c=d(20)+chamod
        return c
```



```
def d(sides):
    roll=int(random.randint(1, sides))
    return roll
```

Fig. 3

Fig. 2

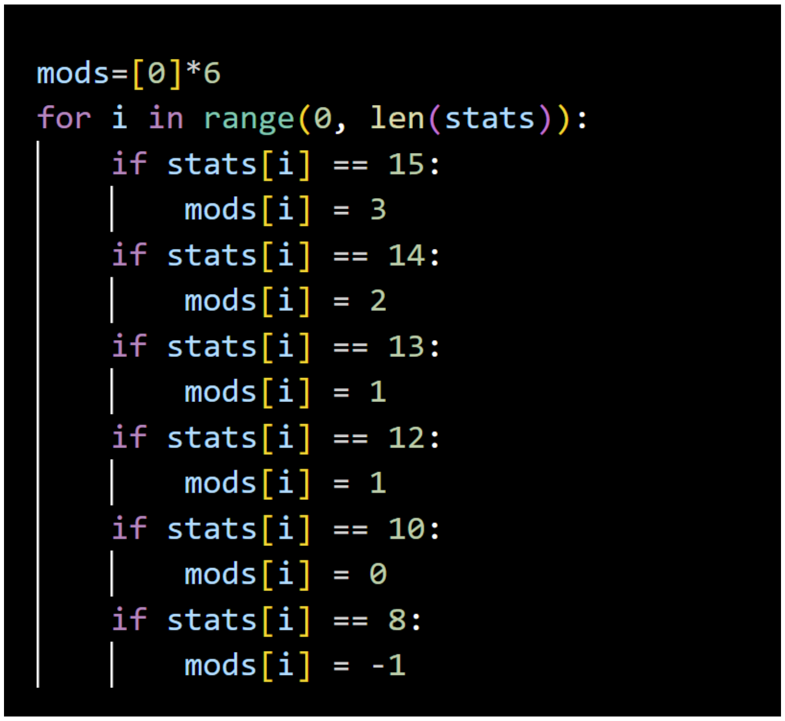
I chose this project because Dungeons and Dragons is a game that I play regularly with my friends and writing these sorts of stories is something that I already do, which made it a natural transition into the project. Dungeons and Dragons is a game that thrives off of the infinite possibilities of the players imagination in the framework of the world they are playing in. While playing at the table with dice, anything you can think of is possible. Obviously, this is not the case in a hard coded videogame environment, so I thought it would be a healthy challenge for myself to create a sufficiently complex environment to explore to still feel like playing a game. It turned out to be a lot of work, but I feel I turned out well for the timeframe I had to work with. I had worked on my other project since September before my unfortunate computer issues, so I feel I was able to put an impressive amount of detail and depth into the game in the month that I had to recreate the project.

Researching for this project involved looking at other text-based adventure games for inspiration, regarding ideas for puzzles/situations to put the players. I was almost completely new to Python before this class, so a great amount of brainstorming was required. I was unable to find examples of similar projects in Python of Dungeons and Dragons/random number based text adventures on the internet, so I had to infer or think of my own implementation techniques for things that I have seen in more mainstream text based games that exist on game store platforms like Steam or things that I have done to players when actually playing Dungeons and Dragons.

Section 2: Design Process

a. Implementation

The way I implemented this project at the most basic level, as talked about above, was through a series of random number generator functions, just like actual Dungeons and Dragons. The first function I created for the project was a function that took in a number, representing the number of sides of the die you wished to roll, and outputs a random number between 1 and the input. This function was then used to create 6 more functions, each one correlating to a specific stat that the player chose, rolling a 20-sided die and then adding the associated bonus. The associated bonuses can be seen below in figure 4.

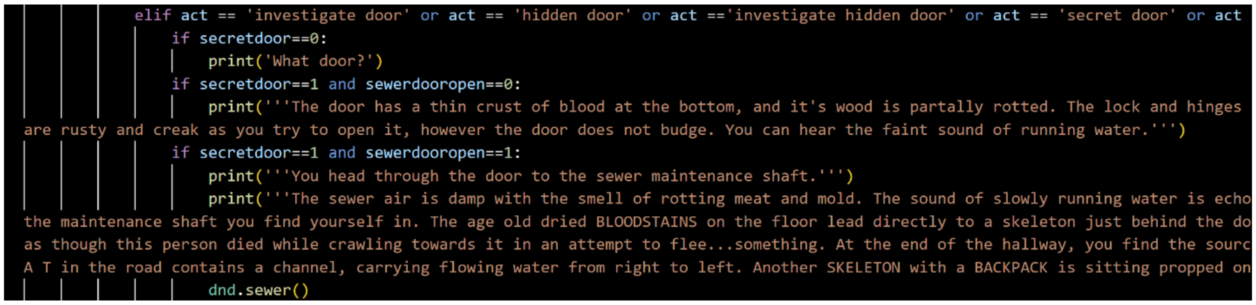


```
mods=[0]*6
for i in range(0, len(stats)):
    if stats[i] == 15:
        mods[i] = 3
    if stats[i] == 14:
        mods[i] = 2
    if stats[i] == 13:
        mods[i] = 1
    if stats[i] == 12:
        mods[i] = 1
    if stats[i] == 10:
        mods[i] = 0
    if stats[i] == 8:
        mods[i] = -1
```

Fig. 4

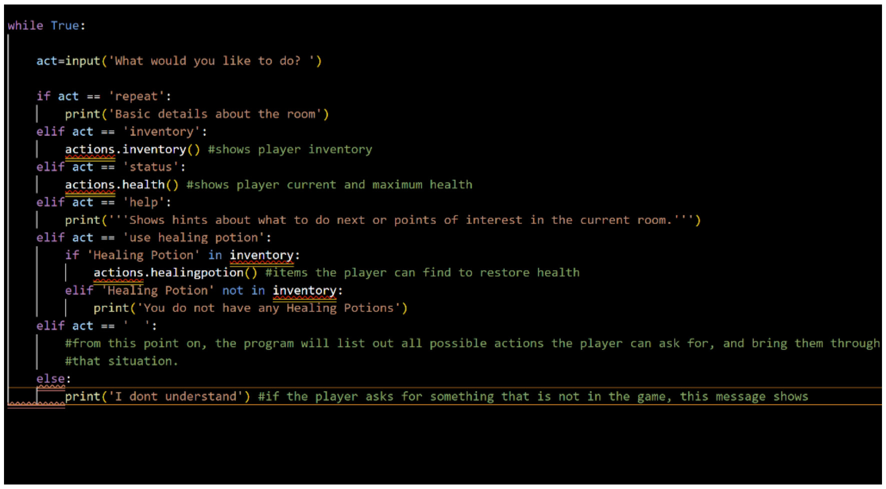
These bonuses, called modifiers, are tacked onto the end of every roll the player makes. This is what makes stat allocation important, as it informs what skills the character will be good at. If you choose to make your Strength stat an 8, you will be receiving a -1 penalty to every roll you make using Strength, meaning you are less likely to succeed at strength-based tasks.

The program itself works on a larger level with a series of classes and embedded functions. The game is broken into 2 classes, ‘actions’ and ‘Dungeons and Dragons’. The actions class contains the stat-specific dice roller functions, while the Dungeons and Dragons function contains the game itself. Each function within the Dungeons and Dragons class represents a room or area of the game, with associated possible checks and situations. A simple example of an area is the ‘Dungeons and Dragons.greatgates()’ function, the first room of the game, which contains a conditionally accessed area that is only available to players that roll high on their initial perception check, which will reveal a hidden door, shown in figure 5 below. Then, within each room/area is a long series of embedded if/then statements inside of a while loop, an example of which shown below in figure 6.



```
                elif act == 'investigate door' or act == 'hidden door' or act =='investigate hidden door' or act == 'secret door' or act
                    if secretdoor==0:
                        print('What door?')
                    if secretdoor==1 and sewerdooropen==0:
                        print('''The door has a thin crust of blood at the bottom, and it's wood is partally rotted. The lock and hinges
are rusty and creak as you try to open it, however the door does not budge. You can hear the faint sound of running water.''')
                    if secretdoor==1 and sewerdooropen==1:
                        print('''You head through the door to the sewer maintenance shaft.''')
                        print('''The sewer air is damp with the smell of rotting meat and mold. The sound of slowly running water is echo
the maintenance shaft you find yourself in. The age old dried BLOODSTAINS on the floor lead directly to a skeleton just behind the do
as though this person died while crawling towards it in an attempt to flee...something. At the end of the hallway, you find the sourc
A T in the road contains a channel, carrying flowing water from right to left. Another SKELETON with a BACKPACK is sitting propped on
                        dnd.sewer()
```

Fig. 5



```python
while True:

    act=input('What would you like to do? ')

    if act == 'repeat':
        print('Basic details about the room')
    elif act == 'inventory':
        actions.inventory() #shows player inventory
    elif act == 'status':
        actions.health() #shows player current and maximum health
    elif act == 'help':
        print('''Shows hints about what to do next or points of interest in the current room.''')
    elif act == 'use healing potion':
        if 'Healing Potion' in inventory:
            actions.healingpotion() #items the player can find to restore health
        elif 'Healing Potion' not in inventory:
            print('You do not have any Healing Potions')
    elif act == '  ':
        #from this point on, the program will list out all possible actions the player can ask for, and bring them through
        #that situation.
    else:
        print('I dont understand') #if the player asks for something that is not in the game, this message shows
```

Fig. 6

Each room function continues to ask the player what they would like to do until they leave the room or die, continually prodding the player to act on the knowledge that they have or to attempt to gather more information.

b. Challenges and Road blocks

Throughout the development of the project, numerous things went wrong that required fixing. In the early stages of my first iteration, the roll variables were generated upon request, spitting out a different number every time. This resulted in the player being able to continually reroll checks until getting a favorable outcome, completely negating the purpose of a skill check. To fix this, I went through my project and placed all of the dice roll functions inside the functions, but outside of the loops, as shown in figure 7 below. Using this, the dice roll variables are predetermined, but hidden from the user. If the user decides to attempt the task, the value is displayed to the user and the outcome from that value is undertaken.

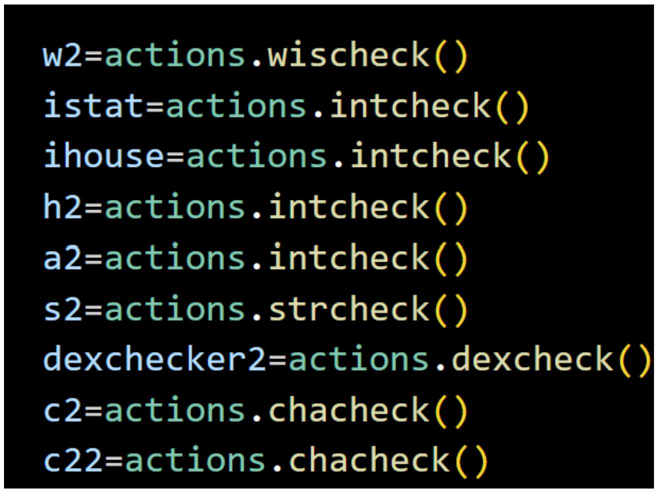

```
w2=actions.wischeck()
istat=actions.intcheck()
ihouse=actions.intcheck()
h2=actions.intcheck()
a2=actions.intcheck()
s2=actions.strcheck()
dexchecker2=actions.dexcheck()
c2=actions.chacheck()
c22=actions.chacheck()
```

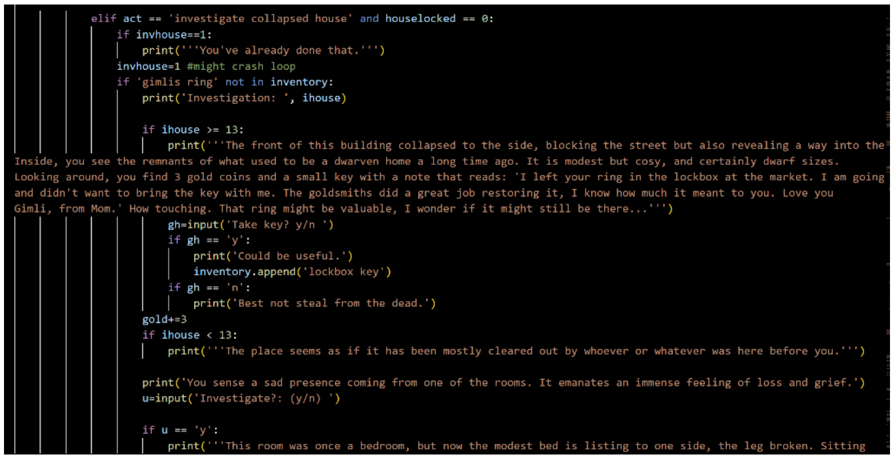
Fig. 7

The set of variables determined by the code in Fig. 6 is inside the function 'Dungeons and Dragons.khirnsquare()', but outside of the while loop that asks for the players desired actions, fixing the variable in place no matter how many times the player attempts the task.

Another challenging speed bump I ran into when programming was the complexity that diaologue trees require. Specifically, the diaologue and questline with the ghost in Khirn Square, Dis. The interactions the player can have with her are complicated and varied, relying on certain items being in the players inventory, previous choices, charisma checks, and the players choices in the conversation. This overall resulted in this one conversation tree taking over 350 lines of code, a piece of which is snown below in figure 8, not including any other prerequisites. In figure 8, the specific portion of programming shown is a series of choices that can be made by the player, each possibly leading to death, being shut out of a potentially helpful questline, or the start of 2 separate quests given by the NPC (Non-Player Character), Dis. These quests were particularly difficult to implement because they are both dependent on the players previous choices in other rooms, as well as being able to leave the area of the quest, and then come back with specific items. To work around this, I used several global variables to track the outcome of their conversation with Dis, as well as implemented an inventory system that used a list of strings to denote items, which was originally simply going to be a set of integers denoting how much of something the player had. Obviously, the integer approach would severely limit what kinds of items I could put in the game, and make it more difficult to add new ones. During the conversation with Dis, the player is given 2 to 4 dialogue choice options at each turn, each one branching to it's own unique outcome. Three final outcomes were possible, Dis becoming enraged with the player and throwing them from the house and potentially killing them, Dis falling into a state of panic upon remembering that her son, the person she wants the player to find, was outside when the city was attacked, and finally Dis believing the player is one of her son's friends, and asking the player for 2 things: retrieving a gold ring that had recently been reforged from the market for her, and telling her son to come home if the player runs into him. Both of these questlines proved challenging to implement in their own right, and since there were two separate quests, this required programming for what would happen to Dis when each was completed, as well as when both were completed.

Some problems I ran into while coding this segment were properly reading the players inventory for items, and keeping the permanent variables constant once the player leaves the conversation and comes back later. I was able to solve this by declaring the values globally both inside the function and outside the function, before the Dungeons and Dragons class even starts at the very beginning of the program. Doing this keeps the function completely constant no matter how many times the player leaves the conversation and comes back.

One more problem I overcame within the Dungeons and Dragons.khirnsquare() function was keeping the manhole variable constant. Coming from the sewer cistern room, you have the option to emerge from a manhole into khirn square. I wanted to make it so that you cannot open the manhole from the surface size, but once you go through the sewer and emerge from the manhole, it is then permanently open. To do this, I had to implement a small if/then check so that the manhole variable did not get reset upon returning, shown in figure 9.

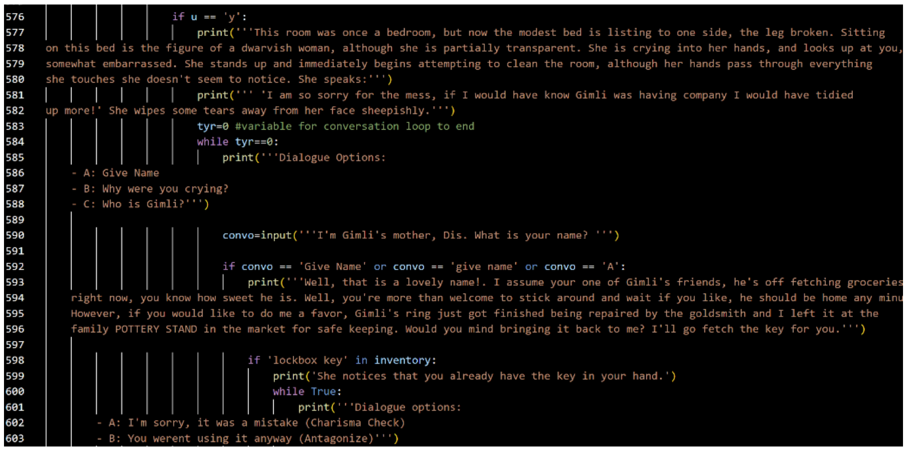


```python
                elif act == 'investigate collapsed house' and houselocked == 0:
                    if invhouse==1:
                        print('''You've already done that.''')
                    invhouse=1 #might crash loop
                    if 'gimlis ring' not in inventory:
                        print('Investigation: ', ihouse)

                        if ihouse >= 13:
                            print('''The front of this building collapsed to the side, blocking the street but also revealing a way into the
Inside, you see the remnants of what used to be a dwarven home a long time ago. It is modest but cosy, and certainly dwarf sizes.
Looking around, you find 3 gold coins and a small key with a note that reads: 'I left your ring in the lockbox at the market. I am going
and didn't want to bring the key with me. The goldsmiths did a great job restoring it, I know how much it meant to you. Love you
Gimli, from Mom.' How touching. That ring might be valuable, I wonder if it might still be there...''')
                            gh=input('Take key? y/n ')
                            if gh == 'y':
                                print('Could be useful.')
                                inventory.append('lockbox key')
                            if gh == 'n':
                                print('Best not steal from the dead.')
                        gold+=3
                        if ihouse < 13:
                            print('''The place seems as if it has been mostly cleared out by whoever or whatever was here before you.''')

                        print('You sense a sad presence coming from one of the rooms. It emanates an immense feeling of loss and grief.')
                        u=input('Investigate?: (y/n) ')

                        if u == 'y':
                            print('''This room was once a bedroom, but now the modest bed is listing to one side, the leg broken. Sitting
```



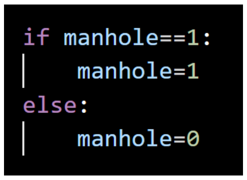
```python
                        if u == 'y':
                            print('''This room was once a bedroom, but now the modest bed is listing to one side, the leg broken. Sitting
on this bed is the figure of a dwarvish woman, although she is partially transparent. She is crying into her hands, and looks up at you,
somewhat embarrassed. She stands up and immediately begins attempting to clean the room, although her hands pass through everything
she touches she doesn't seem to notice. She speaks:''')
                            print(''' 'I am so sorry for the mess, if I would have know Gimli was having company I would have tidied
up more!' She wipes some tears away from her face sheepishly.''')
                            tyr=0 #variable for conversation loop to end
                            while tyr==0:
                                print('''Dialogue Options:
    - A: Give Name
    - B: Why were you crying?
    - C: Who is Gimli?''')

                                convo=input('''I'm Gimli's mother, Dis. What is your name? ''')

                                if convo == 'Give Name' or convo == 'give name' or convo == 'A':
                                    print('''Well, that is a lovely name!. I assume your one of Gimli's friends, he's off fetching groceries
    right now, you know how sweet he is. Well, you're more than welcome to stick around and wait if you like, he should be home any minu
    However, if you would like to do me a favor, Gimli's ring just got finished being repaired by the goldsmith and I left it at the
    family POTTERY STAND in the market for safe keeping. Would you mind bringing it back to me? I'll go fetch the key for you.''')

                                    if 'lockbox key' in inventory:
                                        print('She notices that you already have the key in your hand.')
                                        while True:
                                            print('''Dialogue options:
        - A: I'm sorry, it was a mistake (Charisma Check)
        - B: You werent using it anyway (Antagonize)''')
```

Fig. 8



```python
if manhole==1:
    manhole=1
else:
    manhole=0
```

Fig. 9

This sort of loop is found all throughout the code, and was the only thing I tried that kept the variables constant, despite being a bit ineligant. Implementing this simple check kept the variable constant throughout the game, and prevented the player from going through unless they had already come out the other side first.

c. Iterations

I went through several iterations of this projects final form, both throughout the semester while working on the original, and over winter break when working to redo the project. At the beginning of the semester, the project was very different. Instead of the player choosing stats and allocating them as they chose, the player would instead start by choosing 1 of 3 character classes. These were the Cleric, Rogue, and Fighter. Each of the classes had it's own set of stats, starting equipment, and unique abilities that would separate them from the others. The Cleric gained access to medium armor, which would make them less likely to get hit by attacks, medium health which would allow them to take more damage without dying, and access to magic spells to aid them in their quest, like spells to sense enemies, locate loot, or find a path through the city. The rogue had low health and low armor, making them very weak in open combat, but gained access to the unique abilities to pick locks and sneak around, making it easier for them to take side routes and avoid dangerous situations. They also had the option to attack from stealth, dealing massive damage. The fighter had the most health and toughest armor, but only dealt low damage in comparison to the rogue. Their unique ability was Second Wind, allowing them to heal themselves without needing to find health potions. In the end, both in the interest of time and player enjoyment, I chose to abandon this concept in favor of numerical statistics, and gave the player far more flexibility in their actions. In the older iteration of the game with classes, a player who chose fighter might want to sneak past a group of enemies, but would be unable to, thus making the game feel less fluid. By removing class restrictions on these actions, the player is allowed to act as they wish in all situations, rather than having to make that decision at the very start.

In an early version of the game, after removing class restrictions, the goal of the game wasn't to find a lost archaeologist in the ruins but actually to kill a dragon that was lurking at the bottom. I never began the coding process for this situation, as I decided against it very early on, but while I was storyboarding for the game I decided it would be both thematic and fun for their to be a large final bossfight. In the end, I chose to remove most semblances of open combat from the game and instead focus on problem solving, clue finding, and clever thinking to advance through the game rather than fighting. Additionally, this early version of the game had no NPCs in it, completely removing the need for dialogue or a Charisma stat in favor of a gritty, lone wolf style adventure. This proved not only boring to play, but also boring to code. I thought it would be a fun challenge to implement NPCs into the game that the player could converse with, so in the final version of the game there are 4 total NPCs that the player can talk to. The dialogue sections proved to be some of the most challenging to implement but it was well worth it in the end. The games final storyboard flowchart can be seen belown in figure 10.

The game also had an armor and weapon system in one of the earliest version of the game I had begun to create, which worked much the same as it does in real Dungeons and Dragons. The player would have been gven an Armor Class (AC) score based on their equipment, with heavier armor giving a higher AC at the cost of player dexterity and stealth. Whenever a monster tried to attack the player, they would roll a die to attack, and the numer rolled would have to be equal to or greater than the player's AC, further adding to the fighter's frontline combatant playstyle. I had originally also planned to have the player be able to find and equip dfferent sets of armor and weapons to give individuality to each playthrough, but this was removed very early on due to complications with the program and time restrictions.

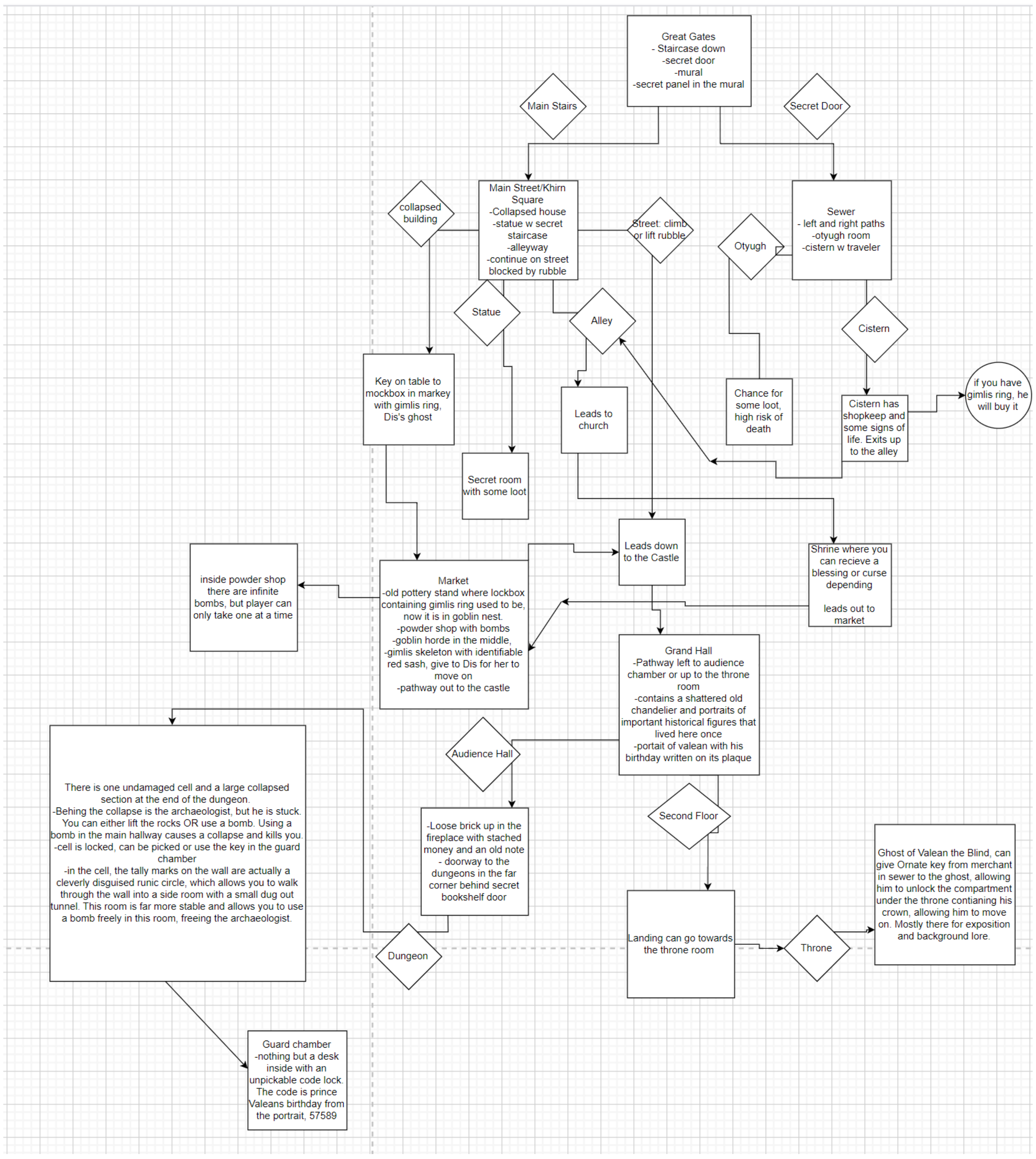

Great Gates
- Staircase down
-secret door
-mural
-secret panel in the mural
Main Stairs
Secret Door
Main Street/Khirn Square
-Collapsed house
-statue w secret staircase
-alleyway
-continue on street blocked by rubble
collapsed building
Street: climb or lift rubble
Otyugh
Sewer
- left and right paths
-otyugh room
-cistern w traveler
Statue
Alley
Cistern
Key on table to mockbox in markey with gimlis ring, Dis's ghost
Leads to church
Chance for some loot, high risk of death
Cistern has shopkeep and some signs of life. Exits up to the alley
if you have gimlis ring, he will buy it
Secret room with some loot
Leads down to the Castle
Shrine where you can recieve a blessing or curse depending
leads out to market
inside powder shop there are infinite bombs, but player can only take one at a time
Market
-old pottery stand where lockbox containing gimlis ring used to be, now it is in goblin nest.
-powder shop with bombs
-goblin horde in the middle,
-gimlis skeleton with identifiable red sash, give to Dis for her to move on
-pathway out to the castle
Grand Hall
-Pathway left to audience chamber or up to the throne room
-contains a shattered old chandelier and portraits of important historical figures that lived here once
-portait of valean with his birthday written on its plaque
There is one undamaged cell and a large collapsed section at the end of the dungeon.
-Behing the collapse is the archaeologist, but he is stuck. You can either lift the rocks OR use a bomb. Using a bomb in the main hallway causes a collapse and kills you.
-cell is locked, can be picked or use the key in the guard chamber
-in the cell, the tally marks on the wall are actually a cleverly disguised runic circle, which allows you to walk through the wall into a side room with a small dug out tunnel. This room is far more stable and allows you to use a bomb freely in this room, freeing the archaeologist.
Audience Hall
-Loose brick up in the fireplace with stached money and an old note
- doorway to the dungeons in the far corner behind secret bookshelf door
Second Floor
Ghost of Valean the Blind, can give Ornate key from merchant in sewer to the ghost, allowing him to unlock the compartment under the throne contianing his crown, allowing him to move on. Mostly there for exposition and background lore.
Dungeon
Landing can go towards the throne room
Throne
Guard chamber
-nothing but a desk inside with an unpickable code lock. The code is prince Valeans birthday from the portrait, 57589

Fig. 10

Section 3: Results

The final result of these efforts is a very long, but efficient program that leads the player through the desired story, presenting them with difficult choices that are often not as simple as they may seem. The player has the opportunity to discover secrets, figure out puzzles, and work their own way through the ruined city in their attempt to accomplish their quest. The text display was difficult to format so that it displayed in a readable manner, and it was also difficult to convey the desired information in a concise and effective manner so that it could be displayed through the terminal. The final product is a satisfactory story game that forces the player to think critically and solve puzzles, find secrets, and complete quests in order to accomplish their overall goal.

Section 4: Concepts Learned

This project proved to be a true challenge to my knowledge of object based programming. I used embedded while loops based on a 'True' declaration as well as based on a variables value, cleverly hidden variables and checks to give the illusion of a procedurally generated environment to the player, assigning values and named within a dictionary to accurately allocate the values of player chosen statistics, took input from the player to influence the program, and used many small functions within the class system of the program to make the game run smoother than it could have otherwise. As stated before, I was almost completely new to python before taking this course. Throughout the semester, learning how to handle objects and classes within a program was one of the biggest challenges for me, but through hard work and research on my own time, I was able to finally come to an understanding of what mistakes I had kept making that was preventing my programs that utilized objects and classes from running smoothly.

Section 5: Retrospective

Looking back at the project, one thing I would have done differently was make it so that leaving an area and then coming back did not allow a player to attempt the same check with different results. This was a design choice I made very early on in development due to how Dungeons and Dragons actually works when played in real life, and I thought would be a fun mechanic to implement, but actually was a bit of a detriment. It would be impossible for me to rework the project away from this direction without completely rewriting large sections of the game, so it was left in, but it does take away from the game in some ways if you can undo bad rolls by simply leaving and coming back.

Another thing I would do if given more time would be to reimplement the class system from the first iterations. Although it was clunky, a bit restrictive to the player, and difficult to program, it truly would have added another layer to the gameplay by increasing the individuality of each players experience. Each of the classes represented a unique playstyle that different players might like to explore. The fighter was a bruiser style melee combatant, while the rogue preferred to avoid combat, and the cleric's utility spell would have been very useful at navigating the portions of the game that took place outside of fighting. More equipment, such as different armor and weapons, for the player to find and utilize would also be something I could add if given more time, which would add another layer of customization to the game. Additionally, adding a more robust combat system revolving around more dice rolls and more player choice would make the game truly spring to life, like was intended in some of the earlier versions of the game.