START SLIDE #1 (TITLE SLIDE)

The problem that we decided to attempt to remedy is the lack of ability to play a Dungeons and Dragons style game in an online medium with a computer based Dungeon Master. Now, this game is still in early development stages, so keep that in mind when you see certain playing components.

SLIDE 2 (SLIDE THAT SAYS COMPONENTS OF CODE)

What we have designed is a text-based adventure, inspired by Dungeons and Dragons. Due to time constraints, we were unable to design it as an exact model of Dungeons and Dragons, but we have taken a page from its book and created our own role-playing adventure. The premise of our game is this:

You're an adventurer from a kingdom that has been ransacked by a series of criminals and your brother who is the King of the kingdom, was kidnapped during the rioting. Your goal is to safely return your brother to his throne, rescuing him from the clutches of the greedy goblins that possess him. Let's take a look at what we've done!

SLIDE3 (THE FIRST FLOW CHART THAT IS FOR THE PARSER)

In order to give the player control over the flow of the game, we needed the program to understand what the player wanted to do. To achieve this, we used Python Lex-Yacc (PLY) which is an implementation of lex and yacc parsing tools for Python. Starting out, we had to define exactly what a word was to the lexer where the definition of a word was a sequence of an unlimited amount of upper and lower case letters before whitespace.

SLIDE4 (LEXER)

From this definition of a word, we then had to create different categories of words, such as nouns, adjectives, and words that simply have the same connotations. These categories are initiated through multiple lists and are then matched by the lexer and added to a list of tokens so the parser can scan the tokens and parse the user's input. As the lexer reads the user's input, the value and type of each word is documented, where the value is the word inputted and the type is the category that the word belongs to (i.e. noun or adjective).

SLIDE5 (PARSER)

As this information is scanned and matched, the parser scans the list of tokens and produces the parsing result based on the specified formatting. Due to the different styles that each player has when it comes to text adventure games, we decided to create a command structure for parsing. That is, as the user's input is scanned it will be associated with a certain command such as an attack command, grab command, use command, etc. This allows the program to

recognize key actions made by the player, making it easier for the program to understand that the player wishes to grab torches, sneak into rooms, or even attack enemies.

Here is an example of how the lexer and parser work together to recognize and execute commands given by the user. In this scenario, there is a dragon blocking your way so you must kill it so you can leave the room.

SLIDE6 (PARSER DEMO)

SLIDE 7 (WEIRD FLOWCHART)

So we designed an overall flow for the game containing the components of data that we needed to persist through the code. We have the overall necessary classes and dictionaries as follows:

-The dungeon map which will be seen with the gui
-Player class that Kaleb will discuss, and
-NPC info (which contains several variables)

SLIDE 8 (NPC DICTIONARY)

Starting with NPCs (non-playable character)… We created a dictionary for the NPC data that contains the information of every NPC listed in the game. The required information ranges from the NPC's name and character strength and HP all the way to the possible items that the NPC is holding that will hold relevance throughout the game. Each item will have certain stats depending on what the item is. It will have some form of strength or defense on it as well as the quantities of the items.

SLIDE 9 (PLAYER DATA)

The player is a vital component of any game and needs to be flushed out completely. That is, the player must have control over their character when it comes to their name, stats, and items. What we have here is but an example of how a player might be set up in the game through a player class. The player data class contains the player's inputted name, it imports random starter stats. We need to know the player's strength, health point amount, how much gold they have, if they have any health potions, and a few other pieces of information. This is then saved into a text file so that it can persist on the occasion that the player wants to save their data and then resume the game at another time.

SLIDE 10 (FLOWCART FOR GAME SCRIPT)

The game file contains every room listed in the map that is functional. It immediately runs a check once the player has entered the room to verify whether or not the player has already been in the room. If they have, the player is then redirected to find a direction they wish to travel.

Once it has been checked and verified that this is the first time it has been entered, the "cutscene" will commence. If there is an opportunity to sneak up on enemies in the room, the player can input what they want. After parsing has commenced, if a sneak-style command has been entered, the player will roll for certain stats that can help them with sneaking up on the enemies. The same will happen with an attack command. As of right now, we have not accommodated certain commands such as seducing with charisma or a bard causing a distraction. Commands like that will be placed under attack or sneak until we expand on the different commands.

Once the enemy NPCs have been neutralized in the scenario, the character is given the opportunity to explore the room that they are in, rifle through the NPCs body to see if they have items on them, and the chance to leave the room in the direction of their choosing. If that direction does not exist according to the layout of the map, the player is notified and prompted again for further input. This process will continue through the game, as this is the primary action sequence for the entirety of the dungeon crawl.

SLIDE 11 (GAME SCRIPT)

Here we have a snippet of the code from the game script consisting of the commands that checks the code to make sure that the room has been checked, if it hasn't, it will go back to the save text file and update the room to show that it has been entered and commences with the gameplay. There are also pieces of the attack sequences and the sneak sequences that checks the user stats, rolls the dice, and checks to see if the player is able to progress accordingly.

The rest of the script is just a reiteration of the flowchart that was seen previously. Each room has the same general structure that is shaken up according to the NPC that can be found in the rooms. Some NPCs will allow for stealth, some are sudden attacks, and some are merchants that possess vital items for progressing through the story.

SLIDE 12 (GUI FLOWCHART)

The part of the code I think all of us are most excited to share would be the GUI itself for the game.

insert GUI info here

The GUI was made with kivy and consists of a main menu, play, instructions, dice roller and map screen. The main menu simply is a screen with two buttons that either take you to play the

game or to a screen with instructions. The instructions screen simply reads off a list of instructions and allows the user to go back to the game. The play screen is the main screen of the game. It has a toolbar, a textbox for the story and a text input for the user to respond with the story. The toolbar has four main functions. It has a back button so that the user can go back to the main menu and help button so the user can view the instructions again. The final button takes the user to a screen that contains a map. Finally, the toolbar also contains the users stats. These stats include the users health or HP, strength and intelligence. Using the text input the player can interact with the story by responding to questions asked by the game.

SLIDE 13 (MAP)

The map screen of the GUI is exactly what it sounds like. It is a picture of a map. The reason that this map is important is because it updates with the game so that the player knows where they are and where they can move. I was able to accomplish this by using a simple sqlite database that contained all of the rooms, the coordinates of the rooms and which rooms were to the north, south, east and west of it. From there I was able to take user input to create sql statements to get the coordinates of the new room. With those coordinates I was able to use matplotlib to plot a dot in the room that the player is located in.

SLIDE 14 (DICE)

The next part of the GUI is the dice roller. In Dungeons and Dragons players need to roll a 20 sided dice to determine the outcome of different paths that they take. For this game we wanted to be able to simulate that happening. Getting a random number is fairly easy, however making it look like a rolling dice was another problem. In order to accomplish this Bethany made 20 png pictures of each side of a 20 sided dice. Each one was named side_#.png so I was able to insert the randomized number into that number and show the dice. To do that I quickly displayed one number while the user rolled the dice and then when the user released the dice the random number would be displayed

SLIDE 15(Connecting All Three Components

The way that the GUI, parser, and game script are connected are through the GUI. The GUI gets the story from the game script which then needs a response. The GUI then gets the user's input and passes it to the parser. Then the parser tells the GUI a simplified version of what the user said. From there the GUI passes the simplified version of the user input to the game script. This process is completed until the end of the game.

Now that we have spent the majority of this presentation discussing the mechanical components of this project, let's take a moment to actually look at the functionality of the game itself.

SLIDE 16 (GAMEPLAY VIDEO)

insert gameplay here