Algorithm and Programming Final Project Report: Abendrot

1. Game Overview

a. Genre

Abendrot is a text-based experimental, adventure game.

b. Game Concept

You play as an adventurer living in a town called Sonnenau. There is no actual set goal in the game, so the player can roam around and talk to any NPCs (Non-playable Characters) to learn more about them and the world.

c. Visual

Being a text-based adventure game, there is little actual visuals to see, but for the few that were implemented (i.e. title screen and scene changes) and the descriptions, it aims for a fantasy setting and feel.

d. Game Flow Summary

The player can go wherever in Sonnenau. In each scene, there can be NPCs the player can talk to. On top of that, like any text-based adventure game, there are also objects that are able to be examined for more information. While there is no actual ending to the game and it's more of a free roam, if the player talks to an NPC named Julian, the game will end.

1. Inspiration

The inspiration from the game came from a variety of sources, namely:

a. Fantasy Novels

From fantasy novels, I took the concept and ideas of an adventurer travelling in a fantasy world full of mythical creatures and magic. Accompanied with the added lore and world-building needed.

Reading through well-written titles can truly immerse a person in an imaginary world, creating connections and emotional attachments that can be very real. So, I wanted to try and take up the challenge of doing just that.

b. RPGs

From RPGs, I took the concept of putting yourself in the shoes of a character that lives in a fantasy world, interacting with NPCs and objects, learning more about the world.

c. Greek, Japanese, and Norse Mythology

Mythology in general has always been a fascinating topic to me. I was always pulled to the intrigue, mystery, and absurdity of the concept of legends, heroes, gods and goddesses. I feel mythology helps us humans to gain a better understanding of how our ancestors viewed the world. It always was fascinating to see that similarities and patterns can be seen in the legends and tales coming from different places of the world.

From there, I chose to focus on three mythoi:

- Greek Mythology
  Found in the form of the constellation names and some place names.
- Germanic and Scandinavian/Norse Mythology
  Found in the form of the legends in the game, place names, NPC names and terms used.
- Japanese or East-Asian Mythology
  Found in the form of a folktale and the flora.

2. Design

a. Objectives

While there are no set objectives, the player is encouraged to explore and interact with the objects and NPCs. Though the game leans to more experimental things and free-roaming, if the player talks to the NPC Julian, the player's attribute of gameOver becomes True, which will end the game.

b. Modules Used

The modules used in the design consist of:

- sys, for stdout
- random, for uniform and choice
- time, for sleep
- art, for text2art
- PIL, for image
- os
- playsound, for playsound

```
from sys import stdout
from random import uniform, choice
from time import sleep
from art import *
from PIL import Image
from playsound import playsound
import os
```

c. General Functions

Stored in: functions.py in core

I used quite a number of functions while creating the code for the game. Their usage ranges from just printing/displaying text, to checking for validation of an input. They are as follows:

- typeWriter, a function to print out text with a typewriter effect.

```
#function for typewriter effect
def typeWriter(text, start=None, stop=None):
    if start is None:
        start = 0.05
    if stop is None:
        stop = 0.1
    for i in range(len(text)):
        stdout.write(text[i])
        stdout.flush()
        sleep(uniform(start, stop))
    return
```

- printBorder, a function to print ASCII border art to add a flourish to the screen when needed, like in the title screen.

```
#function that will print a border for the title, help, credits screens and in general
def printBorder():
    #declaring the border ascii art as the borderArt variable
    borderArt = '''
.--.      .-'.      .--.      .--.      .--.      .--.      .`-.      .--.      .--.      .-'.      .--.      .--.
::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\::::::::.\:::
    `--'      `.-'      `--'      `--'      `--'      `-.'      `--'      `--'      `--'      `.-'      `-.-'      `-
'''
    centerText('\n\n'+borderArt + '\n\n')
```

- printEndcredits, a function to print the end credits screen and play the credits track when the player satisfies the game over requirement.

```
#function to print end credits
def printEndcredits():
    os.system('cls')
    endCredits = text2art("Thank you\nfor playing\nAbendrot.", font='georgia11')
    centerText(endCredits)
    printBorder()
    filename = open('D:\Programming\Github Repos\Final-Project-PDM-2502009646-Rachel_Anastasia_Wijaya\docs\creditsPicture.txt', 'r')
    print(filename.read())
    filename.close
    playsound('D:\Programming\Github Repos\Final-Project-PDM-2502009646-Rachel_Anastasia_Wijaya\docs\The_Koi_and_the_Dragon.wav')
```

- centerText, a function for centering long blocks of text.

```
#function for centering text
def centerText(text):
    for line in text.split("\n"):
        print(line.center(126))
```

- getNPCnames, a function for returning the list of NPC names and aliases in the current scene the player is in, in the form of a list.

```
#function for getting list of npc names in a scene
def getNPCnames(activeScene):
    npcList = []
    for npc in activeScene.npcs:
        npcList.append(npc.name)
        npcList.extend(npc.aliases)
    return npcList
```

- getAllnpcNames, a function for returning the list of **all** NPC names and aliases in the whole game.

```
#function for getting list of *all* npc names
def getAllnpcNames(scenes):
    allNpclist = []
    for scene in scenes:
        for npc in scene.npcs:
            allNpclist.append(npc.name)
            allNpclist.extend(npc.aliases)
    return allNpclist
```

- isNameright, a function for checking if a player's name was entered correctly or not.

```
#function for a while loop that checks if the user typed their name correctly
def isNameright(name):
    while True:
        #prompting user input
        checkName = input(f'\n{name}, is that correct?\n> ')
        #checking if user input is a valid option
        if checkName.strip().lower() in ['no', 'n', 'nope', 'wrong']:
            print('\nI see, please reenter your name.')
            name = isNamevalid(checkName)
            continue
        #stops the loop if no invalid values are found
        elif checkName in ['yes', 'correct', 'y', 'yeah', 'yea', 'yep', 'mhm', 'right']:
            break
        else:
            print('\nI couldn\'t understand that. Try again, little bird.')
            continue
    return name
```

- getPlayername, a function for asking the player for their name as input.

```
#function for collecting the player name from input
def getPlayername(scenes, player):
    #COLLECTING PLAYER NAME
    askPlayername = """Ah, I sense that you are new to this world.\nA lost soul wandering the emptiness of the cosmos.
    \n\nDo tell, what is your name, Vöglein?\n"""
    typeWriter(askPlayername)
    playerName = isNamevalid(scenes, askPlayername)
    playerName = isNameright(playerName)
    typeWriter(f'\n{playerName}... Fascinating.\n',0.3)
    typeWriter('Your name is foreign to my memory, yet the aura you carry within is familiar.\n')
    player.setPlayername(playerName)
    return playerName
```

- isNamevalid, a function to check if the player's name is the same as any NPCs in the game.

```
#function for a while loop that checks if the user's name is the same as any playerNames in the game
def isNamevalid(scenes, playerName):
    while True:
        playerName = input('> ')
        if playerName.strip().lower() not in getAllnpcNames(scenes):
            break
        else:
            print('\nOh? It seems you\'ve the same name as a Sonnenau resident...\nI apologize, but you must pick a different alias.')
            continue
    return playerName
```

- printMap, a function to print out the map of Sonnenau in the form of a .jpg image.

```
#function for printing a visualized map in an image
def printMap():
    #using the open command to open the picture of the Sonnenau map
    sonnenauMap = Image.open("docs/Sonnenau.jpg")
    sonnenauMap.show()
```

- getHelp, a function to print out the game's help screen with instructions and tips.

```
#function to print the help screen
def getHelp():
    printBorder()
    helpText ="""
    This is the essential information needed to play the game.\n\n\n
    >                       Type your commands in the console to do them                    <\n
    >                         Use "map" to bring up the map of Sonnenau                     <\n
    >                    Use "look at" or "examine" to inspect an something                 <\n
    >       Use examine words and 'around' to get a more detailed description of the area   <\n
    >            Use "talk to" or "chat with" to start a conversation with someone         <\n
    >                   Use 'leave' or 'escape' to exit from a conversation                <\n
    >      Use 'help' to bring up these instructions at any point if you forget something  <\n
    >        Use "up", "down", "left", "right" or use cardinal directions to move around   <\n
    >                            Don\'t forget to have fun! ¿ ~^ ˘ ^~ ɔ                      <
    """
    centerText(helpText)
```

d. Input Text Parsing

Stored in: parsertext.py in core

For parsing the commands and input from the player, I used the text parser from “A Christmas Adventure” by Myre Mylar, but with edits here and there, like additional statements to accommodate for the NPC and player interactions. I’ll explain it part by part.
The text parser works by taking the user input, splitting the string into individual words, then using if statements, and checks which command/action the player wants to do. For example, if the user input was “examine book”, seeing examine, the parser matches it with the corresponding examine actions and determines that the player wants to examine something. Then, after confirming which type of action the player wants to do, it will also return the something

- Initial

The first section of the function sets the command and object1 variables to be returned later and the words variable, to be split into a list of individual words.

Then, after checking if the word length is more than 0, sets the remaining words variable and the list of words for different actions the player can do.

```
def parse(input_text):
    command = input_text.lower()
    object1 = None

    # the .split() function splits the input_text string variable into a python list of individual words
    words = input_text.split()

    if len(words) > 0:
        remaining_words_index = 0
        examineActions = ['look', 'inspect', 'check', 'examine', 'study']
        moveActions = ['move', 'walk', 'travel', 'go']
        talkActions = ['talk', 'chat']
```

- Examine Words

The second section of the function focuses on parsing words that are used to examine or inspect objects and NPCs.
Example:

1. User inputs: “examine bookcase”
2. Input text is split: [“check”, “bookcase”]
3. First input text (“check”) is determined to be an examine action, foundExaminewords become True
4. Since there’s only 1 examine word found, the remaining words index is 1
5. Since foundExaminewords is True, the remaining words are processed and returned in the form of a command (“examine”) and object1 (“bookcase”)

```
foundExaminewords = False
if words[0] in examineActions:
    remaining_words_index = 1
    foundExaminewords = True
if words[0] == "look" and words[1] == "at":
    remaining_words_index = 2
    foundExaminewords = True

if foundExaminewords:
    if len(words) > remaining_words_index:
        remaining_words = ""
        for i in range(remaining_words_index, len(words)):
            remaining_words += words[i]
            if i < len(words)-1:
                remaining_words += " "
        command = "examine"
        object1 = remaining_words.lower()
```

- ❖ Move Words

The third section of this function focuses on parsing words that are used to move in the world. Example:

1. User inputs: “go south”
2. Input text is split: [“go”, “south”
3. First input text (“go”) is determined to be a move action, foundMovewords become True
4. Since there’s only 1 move word found, the remaining words index is 1
5. Since foundMovewords is True, the remaining words are processed and returned in the form of a command (“move”) and object1 (“south”)

```
foundMovewords = False
if words[0] in moveActions:
    remaining_words_index = 1
    foundMovewords = True
if (words[0] == 'travel' or words[0] == 'go' or words[0] == 'move' or words[0] == 'walk') and words[1] == 'to':
    remaining_words_index = 2
    foundMovewords = True

if foundMovewords:
    if len(words) > remaining_words_index:
        remaining_words = ""
        for i in range(remaining_words_index, len(words)):
            remaining_words += words[i]
            if i < len(words) - 1:
                remaining_words += " "
            command = 'move'
            object1 = remaining_words.lower()
```

- ❖ Talk Words

The fourth section of the function focuses on parsing words that are used to talk with NPCs. Example:

1. User inputs: “talk to michel”
2. Input text is split: [“talk”, “to”, “michel”]
3. First input text (“talk”) and second input text (“to”) is determined to be a move action, foundTalkwords become True
4. Since there’s 2 move word found, the remaining words index is 2
5. Since foundTalkwords is True, the remaining words are processed and returned in the form of a command (“talk”) and object1 (“michel”)

```
    foundChatwords = False
    if words[0] in talkActions:
        remaining_words_index = 1
        foundChatwords = True
    if (words[0] == 'talk' or words[0] == 'chat') and (words[1] == 'with' or words[1] == 'to'):
        remaining_words_index = 2
        foundChatwords = True

    if foundChatwords:
        if len(words) > remaining_words_index:
            remaining_words = ""
            for i in range(remaining_words_index, len(words)):
                remaining_words += words[i]
                if i < len(words)-1:
                    remaining_words += " "
            command = "talk"
            object1 = remaining_words.lower()

return command, object1
```

e. Characters

Stored in: scene.py in scenes

Characters in the game are created using the Character class. The Character class only has the name attribute and will become the basis for the player character and NPCs.

```
#creating the character class
class Character:
    def __init__(self):
        self.name = ""
```

- ❖ Player Character

The player character is created using the Player class, a subclass of the Character class. It has the attributes gameOver (to check if the game over condition has been met) and name. If I had more time I would’ve added a title attribute for pronouns, but since the deadline’s closing in, I scrapped that idea.

```
#creating the playable character subclass
class Player(Character):
    def __init__(self):
        super().__init__()
        self.gameOver = False

    def setPlayername(self, name):
        self.name = name
```

❖ Non-playable Characters

NPCs are created using the Npc class, also a subclass of the Character class.
It has the attributes:

- aliases, a list to store the possible aliases or nicknames an NPC may have
- dialogueBlurbs, a list to store the pre-conversation blurbs
- dialogueResponses, a list to store the possible responses the player may choose from
- dialogueTexts, a list to store the actual dialogue
- NPCprofile, to store an NPC's profile/backstory for when the player examines them

```
#creating the npc subclass
class Npc(Character):
    def __init__(self):
        super().__init__()

        #aliases, like nicknames or other names for an npc
        self.aliases = []

        #blurbs before initiating dialogue with an npc
        self.dialogueBlurbs = ''

        #dialogue ids, e.g. the player's dialogue response to the npcs in the form of dialogue
        self.dialogueResponses = []

        #dialogue texts
        self.dialogueTexts = []

        #npc backstory and profile
        self.NPCprofile = ""
```

The methods used are as follows:

- addNPCalias, for adding an NPC's alias to the aliases list
- addNPCblurb, for adding an NPC's pre-conversation blurb to the dialogueBlurbs list
- addNPCdialogueResponse, for adding the dialogue responses the player can make when talking to an NPC to the dialogueResponses list
- addNPCdialogueText, for adding an NPC's dialogue text to the dialogueTexts list.

```
#method for adding npc alias to the aliases list
def addNPCalias(self, npcAlias):
    self.aliases.append(npcAlias)

#method for adding the dialogue blurbs before the dialogue choices
def addNPCblurb(self, dialogueBlurb):
    self.dialogueBlurbs = dialogueBlurb

#method for adding the dialogue responses the player can choose
def addNPCdialogueResponse(self, dialogueResponse):
    self.dialogueResponses.append(dialogueResponse)

#method for adding the dialogue texts
def addNPCdialogueText(self, dialogueText):
    self.dialogueTexts.append(dialogueText)
```

f. Scenes

Stored in: scene.py in scenes

Scenes or locations are made using the Scene class. The scene class has the attributes:

- id, for storing a name to identify a scene
- areaName, for storing a scene/area/location's name
- north, south, east, west, each respectively for storing the north, south, east, and west exits of an area
- objects, a list to store the game objects inside an area
- npcs, a list to store the NPCs available in an area

```python
#creating the scene class
class Scene:
    #constructor method for the attributes of the scene class
    def __init__(self):
        #attribute to identify the scene/location
        self.id = ""

        #attribute to store a scene/location/area's name
        self.areaName = ""

        #attributes for the scene exits
        self.north = None
        self.south = None
        self.east = None
        self.west = None

        #attributes for the game objects and npcs in a scene/location
        self.objects = []
        self.npcs = []
```

A note, not all scenes have objects or NPCs to interact with. Some only have objects, with no NPCs.

- The methods used are as follows:
- addObject, for adding a game object to the objects list
- addNPCs, for adding an NPC to the npcs list
- getDescription, getter method for returning the general description of an area in the form of a string
- getExamination,getter method for returning the examination description of an area in the form of a string
- getAreaname, getter method for returning an area’s name

```
    # method to add objects to the objects list
    def addObject(self, object):
        self.objects.append(object)

    # method to add npcs to the npcs list
    def addNPC(self, npc):
        self.npcs.append(npc)

    #getter method that returns a description in the form of a string
    def getDescription(self):
        return ""

    #getter method that returns an examination in the form of a string
    def getExamination(self):
        return ""

    #getter method that returns the area name
    def getAreaname(self):
        return self.areaName
```

g. Game Objects

Stored in: scene.py in scenes

Game objects are created using the gameObject class. The gameObject has has the attributes:

- name, for storing the object's name
- aliases, a list for storing any aliases an object may have

The methods used are as follows

- addAlias, for adding an object's alias to the aliases list

```
class gameObject:
    def __init__(self):
        self.name = ""
        self.detailedDescription = "It's not much to look at."

        #attribute for other names a game object could be called
        self.aliases = []

    #method to add an object's alias to the alias list
    def addAlias(self, objectAlias):
        self.aliases.append(objectAlias)
```

h. Interactivity

All interactivity in the game between the player and the program is done through the console, from text inputs from the player.

- Main Menu

Stored in: abendrot.py

In the main menu, the player will be shown the game title, title screen splash art in the form of a sunset, then a short welcome text to the game, and then a list of main menu options, such as play, help, about, credits, and quit.

Implementation:

1. os.system(‘cls’) is used to clear the console of any text.
2. printBorder() is used to add a border
3. text2art is used from the art module to convert Abendrot into ASCII art form.
4. The function centerText, to center the text and ascii art
5. typeWriter function used to print out a welcome text in an old-timey type-writer style
6. titleScreenoptions function called, used so the player can input their selected option through the console.

```
#declaring a function that will print the title screen
def printTitle():
    #using os functions to clear the console
    os.system('cls')
    #calling the printBorder function
    printBorder()
    #declaring the converted ascii art title to the titleArt variable
    titleArt=text2art("Abendrot", font='fraktur')
    #printing the title art
    centerText(titleArt)
    #declaring the sunset ascii art as the sunset variable
    sunset = '''
                                  ^^                  @@@@@@@@@
                               ^^       ^^           @@@@@@@@@@@@@@@
                                                    @@@@@@@@@@@@@@@@@@              ^^
                                                    @@@@@@@@@@@@@@@@@@@@@
                        ~~~~~ ~~ ~~~~~~ ~~~~~~~~~ ~~ &&&&&&&&&&&&&&&&&&&&&& ~~~~~~~~ ~~~~~~~~~~~~ ~~~
                        ~           ~~~   ~  ~        ~~~~~~~~~~~~~~~~~~~~~~ ~        ~~~      ~~ ~
                          ~       ~~~       ~~ ~~ ~~~  ~~~~~~~~~~~~~~ ~~~~~  ~      ~~~~     ~ ~~~~  ~ ~~~
                          ~  ~~~      ~           ~       ~~~~~~~  ~~ ~~~~        ~~ ~ ~~~  ~~ ~
                        ~  ~        ~ ~       ~            ~~ ~~~~~~~  ~       ~~~  ~              ~~~
                         ~              ~         ~       ~       ~~~   ~              ~
            '''
    #printing the sunset art
    print(sunset)
    #closing the title screen art section by calling another printBorder
    printBorder()
    #printing the welcome text and options for the player
    typeWriter('                                    Welcome to Abendrot, an experimental text-based game.' +'\n'*4)
    titleText = """
    >     PLAY    <\n
    >     HELP    <\n
    >    CREDITS  <\n
    >     ABOUT   <\n
    >     QUIT    <
    """
    centerText(titleText)
    titleScreenoptions()
```

The titleScreenoptions function consists of:

1. A while loop to ask for input until a valid option is entered
2. If statements to check whether the player wants to:
    a. Play the game (“play”), by calling the gameSetup function
    b. Check the help screen for instructions and control (“help”), by calling the helpScreen function

```
#declaring a function that will print the help screen containing the command list/tutorial
def helpScreen():
    printBorder()
    getHelp()
    titleScreenoptions()
```

    c. Check the about screen for info about the game (“about”), by calling the aboutScreen function

```
#declaring a function that will print the screen containing the background behind the game
def aboutScreen():
    printBorder()
    aboutText = """
    This is just extra insight behind the game.\n\n\n
    Abendrot is mostly about telling a story. There's no real end goal, I just wanted to create a
    simple text-adventure game that felt more like an experience, where you just wander around the
    world, talk to people, interact, and learn more about things. It is definitely lacking in several
    technical departments, but I hope it manages to entertain or amuse at the very least.
    """
    centerText(aboutText)
    titleScreenoptions()
```

    d. Check the credits screen for credits (“credits”), by calling the creditScreen function

```
#declaring a function that will print the credits screen
def creditScreen():
    printBorder()
    creditText = """
    These are the credits\n\n\n
    >                    Story idea and characters by Rachel Anastasia Wijaya              <\n
    >                         ASCII art from: https://asciiart.website/                    <\n
    >                       Inspirations, tutorials, and code help from:                   <\n
    >              1. https://www.youtube.com/channel/UC5akxkiQHpxCzPZWskdBbQQ             <\n
    >                    2. https://github.com/MyreMylar/christmas_adventure              <\n
    >             3. https://www.youtube.com/channel/UCnxsHQQIHpJw1ivDgaKp6pA             <\n

    I am also deeply grateful to my close friends, who will remain anonymous,
    who helped me solidify the concept, characters, dialogue, and program. :D
    """
    centerText(creditText)
    titleScreenoptions()
```

    e. Quit the game (“quit”), by using sys.exit to exit the program

```
elif userOption == 'quit':
    #quits the program
    sys.exit()
```

```
#defining the title screen function that will determine the desired user selection
def titleScreenoptions():
    #list of title options in the main menu
    titleOptions = ['play', 'help', 'quit', 'credits', 'credit', 'about']
    #while loop for input validation
    while True:
        #asking for user input
        userOption = input('> ')
        userOption = userOption.strip().lower()
        #checking if user input is a valid option
        if userOption not in titleOptions:
            print('\nPlease enter a valid option from the menu.\n')
            continue
        #stops the loop if no exceptions or invalid values are found
        else:
            break
    #input validation with elif statements
    if userOption == 'play':
        #calls the gameSetup function to start the game
        gameSetup() #will run the game's code

    elif userOption == 'help':
        #calls the helpScreen function to show the help menu
        helpScreen()

    elif userOption == 'credits' or userOption == 'credit':
        #calls the creditScreen function to show the credits screen
        creditScreen()

    elif userOption == 'about':
        #calls the aboutScreen function to show the about screen
        aboutScreen()

    elif userOption == 'quit':
        #quits the program
        sys.exit()
```

- Intro Sequence

Stored in: abendrot.py

If the player types in ‘play’, it will call the gameSetup function which contains the code that will handle the intro sequence. The intro sequence covers:

1. Asking the player’s name with getPlayername function
2. Validating the player’s name with isNameright and isNamevalid functions (inside the getPlayername function)
3. Giving some background/banter before the game starts

```
#### GAME SETUP/INTRO SEQUENCE ####
def gameSetup():
    os.system('cls')
    getPlayername(scenes, myPlayer)
    #INTRODUCTION TO THE GAME
    typeWriter(f"\nYou must have many questions, {myPlayer.name}. I'm afraid I hold none of the answers you seek.\n", 0.1)
    typeWriter("You might find them on this journey. Or perhaps not.\n")
    typeWriter("Nothing in life ever seems to come clear-cut nor unmuddled.\n")
    typeWriter("Philosophical questions and theories stimulate the mind and push us to venture beyond.\n")
    typeWriter("Even if it often leads to existential crises.\n")
    typeWriter("In Sonnenau, perhaps mimesis could be attributed to old tales and legends its folk cling to dearly.\n")
    typeWriter('So come, then, Vöglein. See what you can learn in Sonnenau, from its denizens or even from nature itself.\n')
    typeWriter('Or just, wander around.\nWhatever floats your metaphorical boat, little one.\n')
    typeWriter('Tschüss und viel Glück, Vöglein...\t\t\t\t\t', 0.3)

    #clearing the console to signify the start of the game
    os.system('cls')
    introArt=text2art("Begin...", font='georgia11')
    getHelp()
    printBorder()
    introBlurb = """
    You're an adventurer, hailing from Sonnenau, a coastal town tens of miles away from the mainland
    of Caelis. While being economically reliant on the ocean and the nautical, Sonnenau also prides
    itself of being the second largest hub between Caelis and Iaseon, where merchant boats dock and
    trade; passenger ships transport and ferry people, tourists, adventurers, all sorts; and where the
    annual Festival of the Seafarers is held. You're home now. Just woken up from a (deserved)
    long-night's rest after a fairly taxing quest from Emil, the town's one and only loremaster. Emil
    probably won't mind, if you wait a while, but it might be a good idea to hand over the fruits of
    your labor and hear the sweet, sweet clink of coins in your purse.
    """
    centerText(introArt)
    centerText(introBlurb)
    printBorder()
    gameLoop()
```

- Prompting Action

After the gameSetup function is called, the gameLoop will be called, which contains a while loop that will run as long as the player's attribute of gameOver is False.

If the player's attribute of gameOver is True, it will print the end credits and exit the program.

```
#GAME FUNCTIONALITY
def gameLoop():
    while myPlayer.gameOver == False:
        promptAction()

    if myPlayer.gameOver is True:
        printEndcredits()
        sys.exit
```

While the player attribute of gameOver is False, it will keep calling the promptAction function to prompt the player for an action (move, examine, talk, quit, print map)

The first section of promptAction covers the lists to be compared to, setting the isGameover variable, and setting the activeScene variable as a global variable

```
#function for prompting the player with an input as an action
def promptAction():
    isGameover = False
    global activeScene
    #creating a list of acceptable actions for the player to do
    acceptActions = ['look', 'lookat', 'lookaround' 'inspect', 'check', 'examine', 'move', 'walk', 'go',
                     'travel', 'quit', 'talk', 'chat', 'ask', 'question', 'map', 'help']
    examineActions = ['look', 'lookat', 'inspect', 'check', 'examine']
    moveActions = ['move', 'walk', 'travel', 'go']
    talkActions = ['talk', 'chat']

    #list of valid move directions
    moveDirections = ['north', 'up', 'south', 'down', 'east', 'right', 'west', 'left']

    #list of the character names
    charList = ['mama', 'petra', 'michel', 'julian', 'korvin', 'emil', 'felix', 'ferdinand', 'ingrid', 'nico']
```

The next section covers the while loop which will keep asking the player for what they want to do.

```
#while loop to check if user input is valid
while isGameover == False:
    #prompting for user input, what action they want to do
    playerAction = input('\nSo, what do you wanna do?\n> ')
    parsedAction = parse(playerAction)
```

Below that are if and elif statements to determine if the user input is valid and what action the player wants to do.

1. Checking validity of input

```
#if statement to check if the parsed action is valid
if parsedAction[0] not in acceptActions:
    invalidActionlist = ["\nSorry, no dice, you can't do that, try again.", "\nBuddy, I can't understand that, try again.",
    "\nCould you try again? I can't seem to recognize that command.", "\nTut mir leid, Vöglein, unfortunately you can't do that."]
    print(random.choice(invalidActionlist))
    break
```

2. Checking if the player wants to display the help screen

```
#elif statement to print out the help screen
elif parsedAction[0] == 'help':
    getHelp()
    break
```

3. Checking if the player wants to quit the game

```
#elif statement for if user wants to quit the game
elif parsedAction[0] == 'quit':
    print('\nAight, it was fun. Goodbye, for now.')
    sys.exit()
```

4. Checking if the player wants to print the map of Sonnenau

```
#elif statement for printing the map
elif parsedAction[0] == 'map' or parsedAction[0] in examineActions and parsedAction[1] == 'map':
    printMap()
    break
```

5. Checking if the player wants to examine an object or an NPC

```
#elif statement for if user wants to inspect or look at something
elif parsedAction[0] in examineActions :
    if parsedAction[1] in charList:
        output = examineNpc(activeScene, parsedAction[1])

    else:
        output = examineObject(activeScene, parsedAction[1])
    print(output)
    break
```

6. Checking if the player wants to move/travel

```
#elif statement for if user wants to move
elif parsedAction[0] in moveActions :
    if parsedAction[1] not in moveDirections:
        invalidMovedirectionsList = ["Hmm? I didn't catch that, go where? Try again, please.",
        "Where did you wanna go? Could you try again?", "I didn't recognize that direction, sorry.",
        "Try again please, I couldn't understand which direction you meant."]

        print('\n' + random.choice(invalidMovedirectionsList))
        break
    else:
        result = tryScenechange(activeScene, scenes, parsedAction[1])
        activeScene = result[0]
        output = result[1]
        moveHandler(activeScene, output)
        break
```

7. Checking if the player wants to talk to an NPC

```
#elif statement for if the user wants to talk to an NPC
elif parsedAction[0] in talkActions:
    printDialoguechoices(activeScene, parsedAction[1])
    output = tryForNPCdialogue(activeScene, parsedAction[1], myPlayer)
    dialogue = output
    dialogue = dialogue.replace('playername', myPlayer.name)
    typeWriter(dialogue, 0.05)
    break
```

8. Checking if the player attribute of gameOver is True and breaking the loop if it is

```
#if statement for if the gameOver attribute is True, to break out of the loop
if myPlayer.gameOver is True:
    isGameover = True
    break
```

i. Examine System

For examining or inspecting an object or an NPC, I created the functions examineObject and examineNpc respectively.

1. Examine Object

The function examineObject takes 2 arguments (scene, object_name) and returns a description. The function will then iterate through the objects in the scene and check if any names or aliases match with the object_name inputted and return the object's detailed description if found. If the object_name is invalid, it will return a random description from the invalidObject list.

```
#function to examine an object in a scene
def examineObject(scene, object_name):
    invalidObject = ["\nIt's...something, that's for sure, but you don't know what it is exactly.",
                     "\nYou're not sure what that is.",
                     "\nIt's nothing much to look at, surely?"]
    description = choice(invalidObject)

    if object_name == 'around' or object_name == '':
            description = scene.getExamination()
    else:
        for sceneObject in scene.objects:
            if object_name == sceneObject.name:
                description = sceneObject.detailedDescription

            else:
                for alias in sceneObject.aliases:
                    if object_name == alias:
                        description = sceneObject.detailedDescription

    return description
```

2. Examine NPC

The function examineNpc does the same thing as the examineObject function, only taking the argument npcName instead of object_name, since it's now iterating through the list of npcs in a scene instead of objects.

Same as the examineObject function, it iterates through the list of npcs in a scene and returns their description (NPC profile) and returns an invalid description if a matching npc is not found.

```
#function to return npc profile for if a player examines an NPC
def examineNpc(scene, npcName):
    invalidNpc = ["\nYou're not sure who that is.", "\nThere's not much to say about this person."]
    description = choice(invalidNpc)
    for sceneNpc in scene.npcs:
        if npcName == sceneNpc.name:
            description = sceneNpc.NPCprofile

        else:
            for alias in sceneNpc.aliases:
                if npcName == alias:
                    description = sceneNpc.NPCprofile
    return description
```

j. Dialogue System

For the dialogue system, I decided to make a pretty simple system, no branching paths and just inputting a choice of response and reading through the corresponding conversation with the NPC.

1. printDialoguechoices

This function is used to display the available dialogue choices the player can make. It iterates through the NPCs in a scene. First, it checks if the NPChas any dialogue responses, if not then it breaks the loop, if it does, it continues into a for loop which will iterate and print out the NPC's available dialogue responses.

```
####DIALOGUE####
#function for printing the available dialogue choices for the player
def printDialoguechoices(scene, npcName):
    for sceneNpc in scene.npcs:
        if not sceneNpc.dialogueResponses:
            break
        elif npcName == sceneNpc.name:
            print(sceneNpc.dialogueBlurbs)
            y = 1
            for x in range(len(sceneNpc.dialogueResponses)):
                print(f'{y}. ' + sceneNpc.dialogueResponses[x])
                y += 1
        else:
            for alias in sceneNpc.aliases:
                if npcName == alias:
                    print(sceneNpc.dialogueBlurbs)
                    y = 1
                    for x in range(len(sceneNpc.dialogueResponses)):
                        print(f'{y}. ' + sceneNpc.dialogueResponses[x])
                    y += 1
```

2. tryForNPCdialogue

This function is used to try to initiate dialogue with an NPC.

First, declaring the dialogue and dialogueStarted variable. Then, into a for loop which will iterate over the NPCs in the scene and check if the NPC has dialogue responses available. If no responses are available, it breaks and returns the dialogue variable as is. If an NPC has dialogue responses available, it will break the loop and continue to the next section.

```
#function for trying to start a conversation with an npc if a player puts in a talk command
def tryForNPCdialogue(scene, npcName, player):
    #if statements to check if the intended npc to be talked with is valid
    dialogue = "\nYou can't talk to that person right now.\n"
    dialogueStarted = False
    for npc in scene.npcs:
        if npcName == npc.name:
            if npc.dialogueResponses == []:
                return dialogue
            else:
                break
        else:
            for alias in npc.aliases:
                if npcName == alias:
                    if npc.dialogueResponses == []:
                        return dialogue
                    else:
                        break
```

In this next part, if and elif statements will check for possible scenarios where dialogue won't be able to be initiated.

- ❖ When the npcName inputted is the player's name, me, or myself.
- ❖ When the npcName inputted is not recognized as a character name.
- ❖ When the npcName inputted is not in the active scene.

```
if npcName == player.name or npcName in ['me', 'myself']:
    dialogue = '\nWell, if you wanna talk with yourself, be my guest...\n'

elif npcName == '' or npcName not in getAllnpcNames(scenes):
    invalidTalkList1 = ["I don't know who that is.", "With who? Try again, please.",
                        "I don't seem to recognize that name.", "Oh I've heard that name... They're not in Sonnenau though."]
    dialogue = '\n' + choice(invalidTalkList1)+ '\n'

elif npcName not in getNPCnames(currentScene):
    invalidTalkList2 = ["Sorry, they're not here right now.", "Vöglein, they're not in your general vicinity.",
                        "Aiya, I don't see them anywhere here."]
    dialogue = '\n' +  choice(invalidTalkList2) + '\n'
```

Finally, if it passed the validation, it will turn the dialogueStarted variable to True. If dialogueStarted is True, then it will call the function askFordialogueChoice to prompt the player to input their choice then return the dialogue. Else, it will return the invalid dialogue from the elif statements.

```
#else statement for if npc to be talked with is found to be valid
else:
    dialogueStarted = True

if dialogueStarted:
    dialogue = askFordialogueChoice(scene, npcName, player)
    return dialogue
else:
    return dialogue
```

3. askFordialogueChoice

This function is used to ask for the player's choice from the displayed dialogue responses.

I used a while loop for input validation that uses else if, try except statements, and the getNpcdialogue function. If the player inputs words that indicate they want to end the conversation (e.g. leave, exit) it will return the leaveConversation version of dialogue.If the player enters a valid dialogue choice and no errors come up, it will get the desired NPC's dialogue and return it.

```
#function to ask user input for dialogue choices
def askFordialogueChoice(scene, npcName, player):
    dialogue = "\nThat conversation topic isn't available. Try another one.\n"
    while True:
        dialogueChoice = input("> ")
        if dialogueChoice == '0':
            typeWriter(dialogue, 0.05)
            continue
        else:
            if dialogueChoice not in ['leave', 'exit', 'esc', 'x']:
                try:
                    dialogueChoice = int(dialogueChoice)
                    dialogue = getNpcdialogue(scene, npcName, dialogueChoice, player)
                except IndexError:
                    typeWriter(dialogue, 0.05)
                    continue
                except ValueError:
                    typeWriter(dialogue, 0.05)
                    continue
                else:
                    return dialogue
            else:
                leaveConversation = ["\nYou have left the conversation.\n",
                                     "\nYou think it over and decide to not go over to talk.\n",
                                     "\nOn second thought, you want to do something else.\n"]
                dialogue = choice(leaveConversation)
                return dialogue
```

4. getNpcdialogue

This function is used to get the dialogue for the intended NPC.

It takes 4 arguments, scene, npcName, dialogueId (dialogue choice number), player and returns the dialogue variable.

First, creating the dialogue and dialogueStarted variables and setting the latter to False. Then using a for loop to iterate through the NPCs' names and aliases and if found, it updates the dialogue variable with the appropriate dialogue text and turns the dialogueStarted variable to True to indicate that a conversation has started.

If a conversation did start (dialogueStarted = True), it will then proceed to iterate yet again over the NPCs' name and aliases. Then, like before, if found it will remove/delete the dialogue text and responses that have been played out from their respective lists. Also, if the npcName is found to be Julian or his other aliases, it will then set the player's gameOver value to True, thus ending the game.

```
#function to return the dialogue
def getNpcdialogue(scene, npcName, dialogueId, player):
    dialogue = "\nThat conversation topic isn't available.\n"
    dialogueStarted = False
    for sceneNpc in scene.npcs:
        if npcName == sceneNpc.name:
            dialogue = sceneNpc.dialogueTexts[dialogueId - 1]
            dialogueStarted = True
        else:
            for alias in sceneNpc.aliases:
                if npcName == alias:
                    dialogue = sceneNpc.dialogueTexts[dialogueId - 1]
                    dialogueStarted = True

    if dialogueStarted == True:
        for sceneNpc in scene.npcs:
            if npcName == sceneNpc.name and npcName == 'julian':
                player.gameOver = True

            elif npcName == sceneNpc.name:
                sceneNpc.dialogueTexts.pop(dialogueId - 1)
                sceneNpc.dialogueResponses.pop(dialogueId-1)
            else:
                for alias in sceneNpc.aliases:
                    if npcName == alias and npcName == 'jules':
                        player.gameOver = True
                    if npcName == alias:
                        sceneNpc.dialogueTexts.pop(dialogueId -1)
                        sceneNpc.dialogueResponses.pop(dialogueId - 1)
    return dialogue
```

k. Movement System

Stored in: functions.py in core

Movement in the game is confined to the four cardinal directions, which are:

- North (up)
- South (down)
- East (right)
- West (left)

Using the scene class that I mentioned above, each scene has 4 possible exits. Movement is handled using these functions:

- ❖ getScenefromId

This function takes 2 arguments, an id and the list of scenes. It returns the scene that corresponds with the id that it was given.

```
#function for returning the scene's designated id from the scenes list
def getScenefromId(ids, scenes):
    for scene in scenes:
        if ids == scene.id:
            return scene
```

- ❖ tryScenechange

This function takes 3 arguments, the currently active scene, the list of scenes, and the direction the player wants to go towards.
First, I created the changedScene variable as False to check if a scene transition has taken place. Then the list of directions that are valid (north, south, east, west).

After that, comes the elif statements. These will check whether a scene has an exit in the player's desired direction of movement. If it isn't None, it will then call the getScenefromId function, change the active scene and also the changedScene into True.

If the changedScene value is True, it will return the description for the scene. On the other hand if the changedScene is False, it will return a description that will notify the player that they cannot move in that direction.

```
#function for attempting to change the scene/location if the player puts in a move command
def tryScenechange(activeScene, scenes, direction):
    changedScene = False
    moveNorth = ["north", "up"]
    moveSouth = ["south", "down"]
    moveEast = ["east", "right"]
    moveWest = ["west", "left"]

    if direction in moveNorth:
        if activeScene.north is not None:
            activeScene = getScenefromId(activeScene.north, scenes)
            changedScene = True
    elif direction in moveSouth:
        if activeScene.south is not None:
            activeScene = getScenefromId(activeScene.south, scenes)
            changedScene = True
    elif direction in moveEast:
        if activeScene.east is not None:
            activeScene = getScenefromId(activeScene.east, scenes)
            changedScene = True
    elif direction in moveWest:
        if activeScene.west is not None:
            activeScene = getScenefromId(activeScene.west, scenes)
            changedScene = True

    if changedScene:
        description = activeScene.getDescription()
    else:
        invalidMoveresponse = ["You can't travel in that direction.", "Whoa, slow down, that way's a no-go.",
                               "Ach nein, you can't go that way, sorry.",
                               "If I could let you pass I would, but... I can't so, try again, Vöglein."]
        description = choice(invalidMoveresponse)

    return activeScene, description
```

- ❖ moveHandler

Last of the movement functions is the moveHandler. This function will print out the summary of the attempt at a scene change. If it isn't successful, it will print out that the player can't move in that direction and that they are still in the same scene. If it's successful, it will print out that the player has moved to the new, changed scene. It will also print out the description of the new scene.

```
#function for printing the player's current location
def moveHandler(current_scene, description):
    invalidMoveresponse = ["You can't travel in that direction.", "Whoa, slow down, that way's a no-go.",
                           "Ach nein, you can't go that way, sorry.",
                           "If I could let you pass I would, but... I can't so, try again, Vöglein."]
    if description in invalidMoveresponse:
        print('\n' + description)
        print('\nYou are still at ' + current_scene.areaName)

    else:
        print('\nYou have moved to ' + current_scene.areaName)
        printBorder()
        centerText(current_scene.areaName.upper())
        centerText(description)
```

I. Game Driver

Stored in: abendrot.py

In this section I will explain how the file containing the game driver code works.

First, I imported the modules and classes that I've made.

```
#importing the needed modules
from core.parsertext import parse
from core.functions import tryScenechange, examineObject, getHelp, moveHandler, examineNpc, printDialoguechoices
from core.functions import centerText, typeWriter, printBorder, printMap, getPlayername, tryForNPCdialogue, printEndcredits
from art import *

import random
import sys, os

#importing the individual scenes from the scenes folder
import scenes.scene as char
import scenes.scene1 as s1
import scenes.scene2 as s2
import scenes.scene3 as s3
import scenes.scene4 as s4
import scenes.scene5 as s5
import scenes.scene6 as s6
import scenes.scene7 as s7
import scenes.scene8 as s8
import scenes.scene9 as s9
import scenes.scene10 as s10
import scenes.scene11 as s11
```

Then I created a list to store and initialize the scene classes. Created the activeScene variable and assigned it to scene_1 (the start scene). Set the display size for the command prompt window. Also, using the getAllnpcNames I stored all NPC names and aliases in the list charList. Finally I created the player character by creating the myPlayer variable and initializing the Player class.

Below this are the functions:

- promptAction
- gameLoop
- titleScreenoptions
- printTitle
- helpScreen
- aboutScreen
- creditScreen
- gameSetup

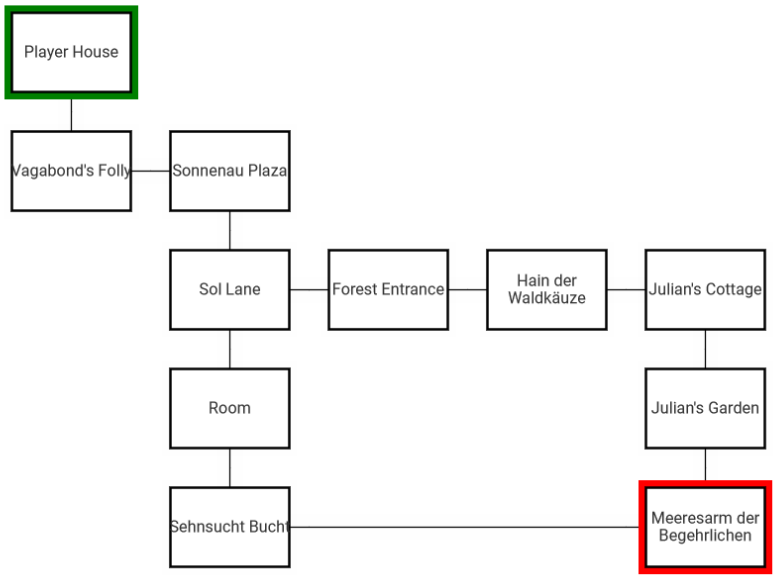
```
#creating a list for the scenes in the game
scenes = []
scene_1 = s1.SceneOne()
scenes.append(scene_1)
scenes.append(s2.SceneTwo())
scenes.append(s3.SceneThree())
scenes.append(s4.SceneFour())
scenes.append(s5.SceneFive())
scenes.append(s6.SceneSix())
scenes.append(s7.SceneSeven())
scenes.append(s8.SceneEight())
scenes.append(s9.SceneNine())
scenes.append(s10.SceneTen())
scenes.append(s11.SceneEleven())

#setting the display size of the cmd window
os.system("mode con cols=130 lines=45")

#setting the initial active scene as the first scene
activeScene = scene_1

#creating the player
myPlayer = char.Player()

#getting the entire character and alias list
charList = getAllnpcNames(scenes)
```

Finally, way down below is where I call the printTitle function to start the game when the program is run.

```
abendrot > abendrot.py > ...
269     typeWriter('So come, then, Vöglein. See what you can learn in Sonnenau, from its denizens or even fro
270     typeWriter('Or just, wander around.\nWhatever floats your metaphorical boat, little one.\n')
271     typeWriter('Tschüss und viel Glück, Vöglein...\t\t\t\t\t', 0.3)
272
273     #clearing the console to signify the start of the game
274     os.system('cls')
275     introArt=text2art("Begin...", font='georgia11')
276     getHelp()
277     printBorder()
278     introBlurb = """
279     You're an adventurer, hailing from Sonnenau, a coastal town tens of miles away from the mainland
280     of Caelis. While being economically reliant on the ocean and the nautical, Sonnenau also prides
281     itself of being the second largest hub between Caelis and Iaseon, where merchant boats dock and
282     trade; passenger ships transport and ferry people, tourists, adventurers, all sorts; and where the
283     annual Festival of the Seafarers is held. You're home now. Just woken up from a (deserved)
284     long-night's rest after a fairly taxing quest from Emil, the town's one and only loremaster. Emil
285     probably won't mind, if you wait a while, but it might be a good idea to hand over the fruits of
286     your labor and hear the sweet, sweet clink of coins in your purse.
287     """
288     centerText(introArt)
289     centerText(introBlurb)
290     printBorder()
291     gameLoop()
292
293 printTitle()
294
295
```

3. Level Design (Creating a Scene)

In this section I will be explaining how a scene in the game is designed. I will only give one example, for brevity and because it would take up around 50 pages (maybe even more?) just to finish explaining each and every scene.



But for reference, the entire map is connected as such:

The green bordered room is where the player starts from and the red bordered room is where the player ends the game if the right conditions are met.

Stored in: scene11.py in scenes

I will pick the scene that contains the NPC that will trigger the ending of the game if talked with. The scene/location is named the “Meeresarm der Begehrlichen” but that’s a mouthful so I will refer to it as the inlet for ease.

First up, I imported the needed classes and initialized a subclass of the Scene class named SceneEleven (in accordance to the scene’s order/id).
First I set :

- The area name as “Meeresarm der Begehrlichen”
- The scene’s id as “scene11”
- The scene’s available exits as north (“scene10”) and west (“scene6”)

```
'''
Setting the class for scene 11, or the Meeresarm der Begehrlichen, inherited from the scene class.

'''

from scenes.scene import gameObject, Scene, Npc


class SceneEleven(Scene):
    def __init__(self):
        super().__init__()
        #setting area name
        self.areaName = "Meeresarm der Begehrlichen"

        # setting id
        self.id = "scene11"

        # creating exits
        self.north = "scene10"
        self.west = 'scene6'
```

Next, I started working on the available objects in the scene. I set:

- The sunset object as a gameObject()
- The shore object as a gameObject()

With each object, I set their names, aliases(if there are any), and a detailed description. Then I added each object to the objects list in the scene.

```
# creating objects
sunset = gameObject()
sunset.name = 'sunset'
sunset.addAlias('setting sun')
sunset.addAlias('red sun')
sunset.detailedDescription = """
You don't know why, but sunsets in the inlet always seem a bit different from one viewwed at the
Sehnsucht Bucht. It's odd, but it lends a certain air of individuality that makes the inlet just
that much more special.

At least, through your eyes.
"""
self.addObject(sunset)

shore = gameObject()
shore.name = 'shore'
shore.addAlias('beach')
shore.addAlias('strand')
shore.addAlias('sand')
shore.addAlias('sands')
shore.addAlias('shoreline')
shore.addAlias('seashore')
shore.detailedDescription = """
The shoreline here is calm. The waves don't really come in droves and you never seem to find any
swept up debris, sealife, or reeds.
"""
self.addObject(shore)
```

After finishing up with the objects, I proceeded to create the NPCs–or NPC in this case–that will be present in the scene. Here, it's Julian, so I set:

- Julian as an Npc()
- Added Julian's name
- Added Julian's aliases
- Added Julian's NPC profile

Then I added the Julian NPC to the npcs list in the scene.

```
# creating npcs
#### JULIAN ####
julian = Npc()
julian.name = 'julian'
julian.addNPCalias('jules')
julian.NPCprofile = """
Julian. What more can you say about him? Same like Nico, he doesn't like speaking much about
the past. Though... compared to the former, he is more open when the topic actually comes up.
Probably more of a you're on a need-to-know basis, if it's not brought up, no reason to trudge
down memory lane.

Before moving out to the woods, he used to live in Sonnenau, near the plaza. But, for some reason
he hasn't shared with you, he decided to uproot his life there and live a hermit's life.

Other than that, it's not official, but he's become sort of Sonnenau's resident herbalist and doctor.
Growing a personal garden has its uses, y'know.

Whenever your busy schedule and his... uh.. more liberal one allows, you like to meet up in the inlet
just south of his cottage. It's nice to unwind with a friend, take the adventuring gear off and just...
Talk.
"""
self.addNPC(julian)
```

After that, I could finally start adding the dialogue for Julian. Since I was closing in to the deadline, Julian (and most of the other NPCs} have only 1 dialogue option. So I added:

- The dialogue blurb for Julian
- The dialogue text for Julian
- The dialogue response for Julian

```
#### JULIAN DIALOGUE ####

julianBlurb = """
You see Julian sitting on the sand, closer to the rippling waves than dry land. He's facing the sunset.
But as he hears the soft, but audible crunch of footsteps on sand, he turns.
"""

julian.addNPCblurb(julianBlurb)

julianDialogueText1 = """
His silhouette, against the still lit sky, creates a chiaroscuro as vivid as one would see in an
artist's meticulously crafted masterpiece.

With a smile as bright as the rays that shadow his face, he says:

"Oh. playername. I've been waiting for you. Come, take a seat."

"""

julian.addNPCdialogueText(julianDialogueText1)
julian.addNPCdialogueResponse(""Jules? Hey."")
```

And we're done with the NPCs and objects, so now all that's left to do is set the scene's general description and its examination description. Using the getDescription and getExamination methods, I added the needed text and voila! A scene has been completed.

```
def getDescription(self):
    description = """
    The Meeresarm der Begehrlichen, another name for the Inlet of the Desiring. A secluded
    inlet that connects to the Sehnsucht Bucht to the east. It's just a short stroll from
    Julian's abode, which is why while not fully private from prying eyes, it's become your
    go-to place to meet and hang out with Julian. Also, the name certainly invokes some vivid
    imagery. Surely there's a story behind it?
    """

    return description

def getExamination(self):
    examination = """
    Somehow, every time you visit this tiny little inlet, it always manages to stun you to
    silence. You attribute it to a spell the inlet puts visitors under, but Julian--more
    attuned to magic than you are--denies it. This time is no different. It looks, for lack of
    a better or more eloquent word, stunning. The way the sun meets with the sea's horizon,
    reflecting and refracting back and forth like a stroke of a paint brush.

    The afterglow.

    Oh? You can just roughly make out Julian's figure on the distance.
    """
    return examination
```

4. Art and Story Concepts

In this section I will be giving a bit of background about the game's concepts.

a. Story

This is what the player is greeted with after they finish the intro sequence. It intends to give a little bit of backstory and set up the general premise/setting of the game.

"You're an adventurer, hailing from Sonnenau, a coastal town tens of miles away from the mainland of Caelis. While being economically reliant on the ocean and the nautical, Sonnenau also prides itself of being the second largest hub between Caelis and Iaseon, where merchant boats dock and trade; passenger ships transport and ferry people, tourists, adventurers, all sorts; and where the annual Festival of the Seafarers is held. You're home now. Just woken up from a (deserved) long-night's rest after a fairly taxing quest from Emil, the town's one and only loremaster. Emil probably won't mind, if you wait a while, but it might be a good idea to hand over the fruits of your labor and hear the sweet, sweet clink of coins in your purse."

I admit that it's probably a pretty generic setup and setting, but I always wanted to write something fantasy themed, so I thought this was the perfect opportunity for it. I wanted to make sure that the player wasn't some world-renowned hero, but also not nobody, just a run-of-the-mill adventurer who happens to have some downtime after an adventure. Though it might not be everyone's cup of tea, seeing as the game is mostly just wandering, examining objects, and talking to NPCs, lacking any thrills or typical adventuring, I hope that it still manages to amuse or entertain some people.

b. Map and World

For the world that the game is set in, again I was inspired by the fantasy genre.
These are a list of the locales (with short descriptors) that are referenced in the game:

- Feoyioran, a realm/continent, advanced
- Caelis, an advanced country, mainland from Sonnenau
- Sonnenau, a coastal town
- Iaseon, an advanced country to the east
- Ocrus, a city renowned for its academies in Caelis
- Euzes, ancient town north of Sonnenau, where owls used to nest
- Hull, a mining town in Caelis
- Eschenroden, a conservative monarchy in Caelis, revolution is seemingly close
- Mugen no tochi, a town in Iaseon

Focusing on Sonnenau, it is a town inspired by Germanic and Northern European towns.

For creating the basic map, used for planning the rooms and how they connect, I used http://trizbort.io/

For creating the fully fledged map, I used https://inkarnate.com.

c. Characters

The characters mostly have names that are heavily inspired by Germanic names and surnames, since the setting is in a Germanic inspired town.
Here is the list of characters and the location in which they appear:

- Petra, Vagabond's Folly
- Michel, Vagabond's Folly
- Julian, Inlet (Meeresarm der Begehrlichen)
- Emil, Julian's Garden
- Felix, Sonnenau Entrance
- Ingrid, Sonnenau Plaza
- Korvin, Sonnenau Plaza
- Bard(Ferdinand von Albrecht), Vagabond's Folly
- Nico, Forest (Hain der Waldkäuze)

d. Audio

The audio and sound design for the game isn't fleshed out, due to the lack of time. So I only managed to add audio for the end credits scene. Since I had a past composition of mine lying around, I decided to use it so as to not need to worry about royalties or any other complications.

The track, along with its music sheet can be found here: [The Koi and the Dragon]

The audio is implemented in the printEndcredits function using the playsound module to play the audio.

```
#function to print end credits
def printEndcredits():
    os.system('cls')
    endCredits = text2art("Thank you\nfor playing\nAbendrot.", font='georgia11')
    centerText(endCredits)
    printBorder()
    filename = open('D:\Programming\Github Repos\Final-Project-PDM-2502009646-Rachel_Anastasia_Wijaya\docs\creditsPicture.txt', 'r')
    print(filename.read())
    filename.close
    playsound('D:\Programming\Github Repos\Final-Project-PDM-2502009646-Rachel_Anastasia_Wijaya\docs\The_Koi_and_the_Dragon.wav')
```

e. Art

As for the art, the only art that I use are for the title screen, intro sequence, and the end credits. Though I did use the border art as needed in the game.

For the title screen, I used a sunset and border ASCII art and the text2art function from the art module.

- Sunset ASCII art can be found here:[Christopher Johnson's ASCII Art Collection - Sunset - Sunrise - Horizon]
- Border ASCII art can be found here:[Christopher Johnson's ASCII Art Collection - Border]
- Text2art font used is fraktur

```
fraktur :
    .....                .       .           ...            .....
  .H8888888h.  ~-.     x88f` `..x88. .>     .x888888hx    :  .H8888888h.  ~-.        oe      .--~*teu.      .x~~"*Weu.
 888888888888x  `>  :8888   xf`*8888%    d888888888888hxx   888888888888x  `>     .@88     dF     988Nx    d8Nu.  9888c
X~     `?888888hx~ :8888f .888  `"`     8" ... `"*8888%`  X~     `?888888hx~ ==*88888    d888b   `8888>  88888  98888
'      x8.^"*88*"  88888' X8888. >"8x  !  "   ` .xnxx.    '      x8.^"*88*"     88888    ?8888>  98888F  "***"  9888%
 `-:- X8888x       88888  ?88888< 888> X X   .H8888888%:   `-:- X8888x          88888     "**"  x88888~       ..@8*"
      488888>      88888   "88888 "8%  X 'hn8888888*"   >       488888>         88888         d8888*`     `````"8Weu
    .. `"88*       88888 '  `8888>     X: `*88888%`     !     .. `"88*          88888        z8**"`   :   ..    ?8888L
  x88888nX"      . `8888> %  X88!      '8h.. ``      ..x8>   x88888nX"      .    88888      :?.....  ..F :@88N   '8888N
 !"*8888888n..  :   `888X  `~""`   :    `88888888888888f   !"*8888888n..  :     88888     <""888888888~ *8888~  '8888F
'    "*88888888*      "88k.      .~      '%8888888888*"   '    "*88888888*      88888     8:  "888888*  '*8"`   9888%
       ^"***"`         `""*==~~`          ^"****""`            ^"***"`    '**%%%%%%**  ""    "**"`     `~===*%"`
```

For the intro sequence, I used the text2art function and the ASCII border art.

- Same as before, the border art can be found here: [Christopher Johnson's ASCII Art Collection - Border]
- Text2art font used is georgia11

```
georgia11 :


MMP""MM""YMM `7MM"""YMM    .M"""bgd MMP""MM""YMM
P'    MM    `7   MM     `7  ,MI     "Y P'    MM    `7  __,
      MM         MM    d     `MMb.           MM       `7MM   pd*"*b.  pd""b.
      MM         MMmmMM       `YMMNq.       MM         MM  (O)   j8 (O)  `8b
      MM         MM    Y  , .      `MM       MM         MM       ,;j9      ,89
      MM         MM      ,M Mb      dM       MM         MM    ,-='       ""Yb.
    .JMML.     .JMMmmmmMMM P"Ybmmd"      .JMML.     .JMML.Ammmmmmm         88
                                                                (O)   .M'
                                                                 bmmmd'
```

Finally, for the endCredits, I used the tex2art function, the border art, and a picture from a game I particularly liked that I converted to ASCII art using this website: [Convert images/pictures to ASCII art online! (HTML/text)]

- Again, border art can be found in the same link as before here: [Christopher Johnson's ASCII Art Collection - Border]
- Text2art font used is georgia11, same as the intro sequence
- Picture used can be found here: https://i.pinimg.com/originals/7f/02/6a/7f026ac2472ddecdbc85431689e134f0.jpg