History Story Text Based Adventure Game Coursework

Rigamortus

Analysis

Foreword:

I have always wanted to create a historical game that allows you to truly immerse yourself in the day-to-day decisions of the time and place it is based on. This project not only allows me to use my expertise in programming in python to help materialise this notion but also teaches me the intricacies of game development, ranging from validating inputs using exception handling, to handling the complex use of an interface to allow multiple users to create a save and load system.

Background:

History lessons can be quite theoretical, you can't truly experience the dictatorial rule or experience the complexities of planning a war without being there and scientists are yet to create time machines. So, the aim of my game, Rigamortus, is to have the best and most accurate simulation of time itself. It takes the user on a historical journey providing them with fickle choices that pave their paths. The game not only introduces them to choices but engages the user to participate and pay attention to key details of the story as they might be questioned throughout the game on various levels through various topics on various difficulties and their ability or inability to answer these questions or complete the challenges would result in different scenarios, inevitably making the game an ever-changing labyrinth of choices and decisions.

Current Solutions:

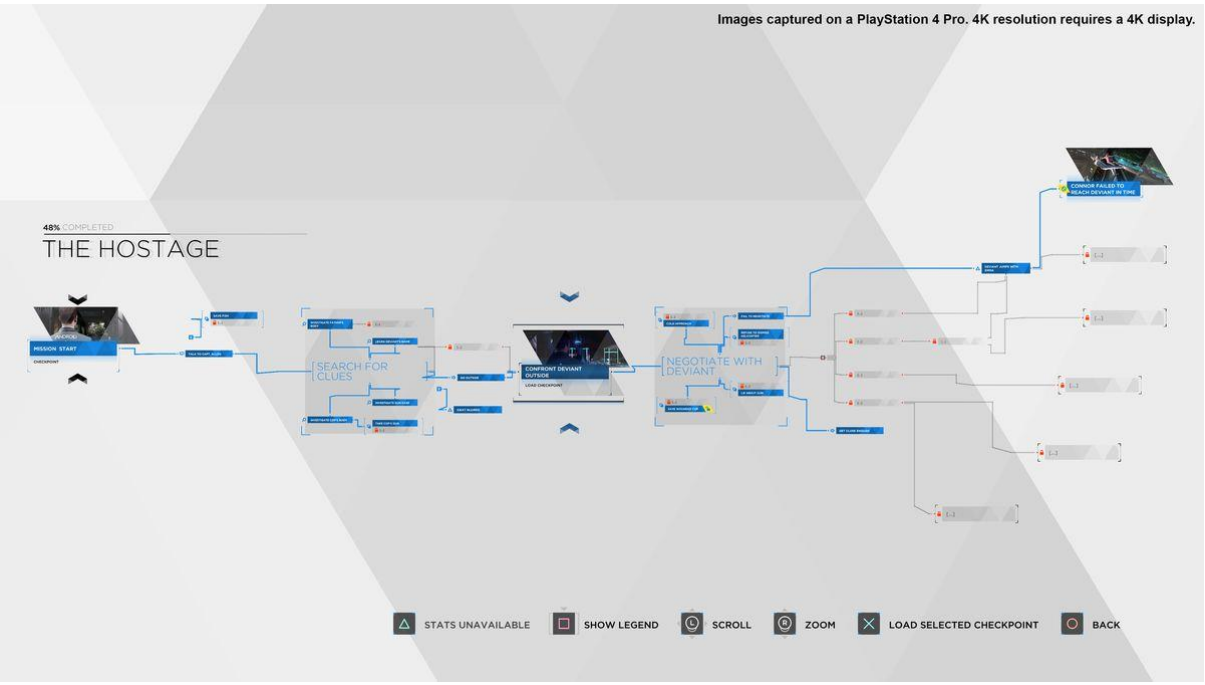
Historical games or even games that take part in some parts of history are either boiled down to building vast arrays of armies to lay siege on a city or objective games with a linear storyline that don't have the flexibility to adapt to the users' choices or needs. My game intends to create a mix of current story-based games as well as decision games with a historical context. Many current story games are notorious for having linear storylines in which no matter what action the user does the end result is still determined and doesn't change which could be quite boring as the user is given an infinite amount of tries to do whatever they want and the consequences of these decisions are never punishing as the user can just reset the level.

A favourite of mine is a real time strategy game called Rise of Nations, which allows you to emulate strategical methods by selecting certain factions, allowing you to try to invade and conquer neighbouring territories, even though this is a pretty accurate simulation of historical timelines its restricted to controlling masses and has a linear lose or win objective

Research:

One of my greatest inspirations for this project was the game - Detroit Become Human. The mechanics of this game heavily appealed to me, especially the decision tree that is displayed in the main menu displaying all the choices you have made throughout the game.

The mechanics of this game inspired me to create systems that extend beyond just the elementary structure of nested if statements and use Object oriented programming and composition to change small user attributes which in turn changes the next step of the game the user would play. However even when it comes to this game despite the variety of options the user is limited to the options provided by the game and are not free to do or more importantly say what they want which can affect the outcome of the game. I have decided to make conversing with the NPC's a major component of my game which in turn adds more flexibility to the game.



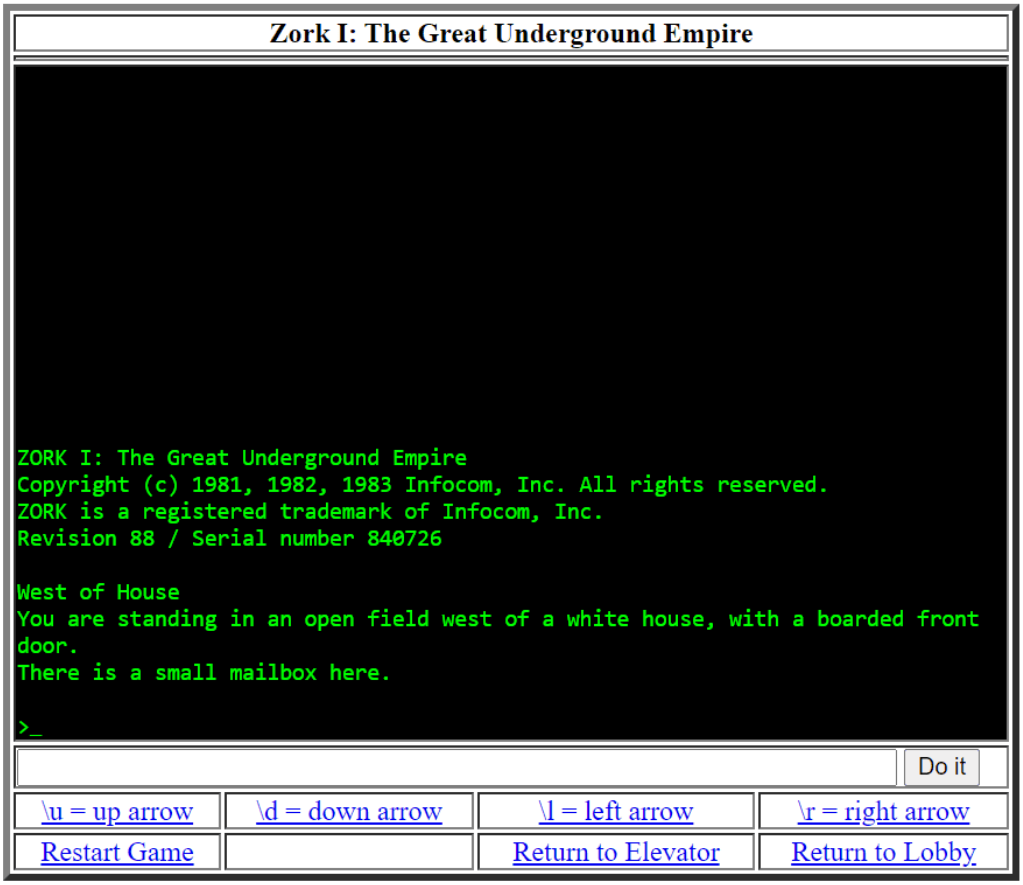
Old School Text Based Adventure Games:

The majority of my game and my user interface is based on classic Text Based Adventure Games such as:

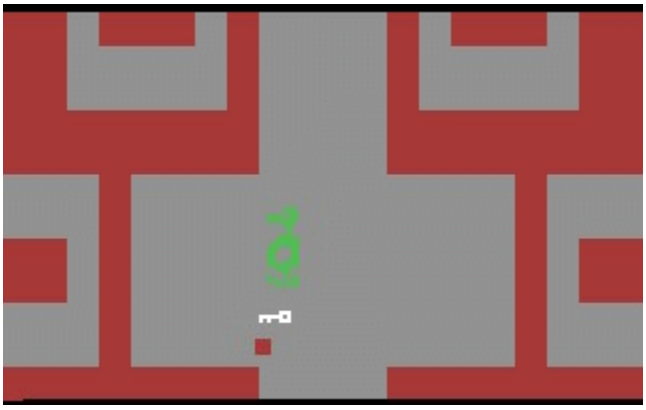
Zork

This game has been my inspiration for the room transition mechanic in my game which allows players to easily move from one room to another
The navigation of the rooms could be implemented using complex Object-Oriented techniques such as aggregation and composition where one room object is linked to another using a dictionary. This is done using the room link method and is implemented using the direction as the key and the actual room object as the value to the key

Text based games are notoriously difficult to program in terms of validating inputs as I have to make sure I don't reject valid inputs but still don't let the user bypass the system with an invalid or inappropriate input, extreme inputs such as blank answers are often punished with prompts or negative attributes assigned to the user which the user will find out after the current user current statistics method is displayed at the end of the major interaction

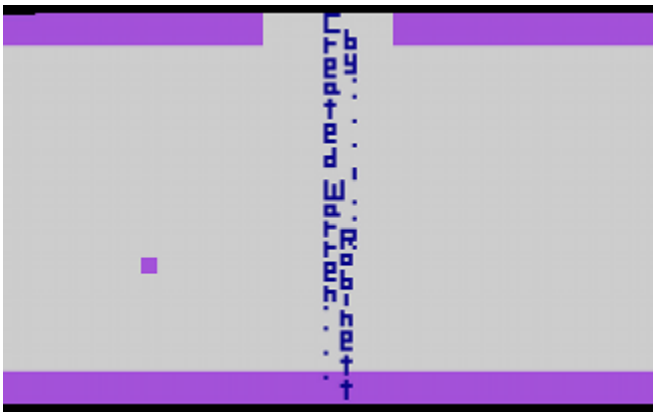


Adventure (1980):



During the 1980s a game developer by the name of Warren Robinett created the first game with an Easter egg, Adventure. During the time game companies did not give credit to their developers nor allowed them to display their names. Warren Robinett however created an easter egg of sorts which consisted of retrieving a key in the middle of the maze, leading you to a room which displayed his name. I was heavily inspired by this to create my own little easter eggs for keeping historical accuracy or even straying away from the real events as long as it is within reason

This adds to a user objective of allowing valid and accurate historical answers to be validated by the game and not ignored. This could be done using nested if statements or using list operations.



Objectives and Goals of the program:

Conversation with the potential end user:

Upon interviewing 2 of my friends, Aarya and Aamir, the end user requirements became more apparent to me that there were many concerns regarding the progression of the game at each level and if they had to restart the entire game in order to look at different combinations and decisions of the game.
They also wanted different logins as well as a system in order to save the data where they left the game last. Many fanatics wanted mechanics that rewarded exploring the outer reaches of the game and exploring beyond the scopes of how the game ideally should be played. An example of this is mentioning the word blitzkrieg during a conversation with one of the NPCs (Non-Player Characters), The General, true history enthusiasts would know this reference and would appreciate the small easter egg.

Prospective Users:

My program is aimed at History majors in aid of simulation or even students enthusiastic about the past. My game and its details pertaining to the mechanics and the background have to be relatively accurate as it would be as much of a simulation as a game.

User needs and objectives:

1) General:

1) All the files needed for the program should be provided with it
2) Game should be able to run on command prompt for any os system
3) User should be able to understand what is expected of them for input
4) Game should run without any bugs or errors of sorts

2) Story Progression:

1) Story should be easy to follow and then user should be able to navigate the rooms easily
2) The user should be able to travel to other rooms and engage with the story but still be prompted to complete the storyline in order
3) The user should be able to interact not only with the NPCs set for each room but with the room itself
4) The user should be rewarded for specific answers that change the due course of their storyline and should be notified about it

3) Minigames:

1) The minigames should load immediately when called and should close either when the game has finished or if the user decided to quit early
2) If the user decides to quit early the game should still return a score and not crash
3) The maze game should be able to use a recursive algorithm to give the user an opponent
4) The algorithm should follow either breadth first, depth first or a random graph traversal method depending on the users' attributes
5) Use the time library to keep track of time for mini games with a time limit

4) Room and Characters:

1) The use of inheritance to efficiently classify the main boss from his subordinates as this would allow the subordinates to have attributes such as loyalty which can be attested and changed but has no relevance to the main boss
2) Linking the room class with the item class using linked list operations to allow items to be randomly assigned to a location in the room inventory and allow the user to search the room to find it
3) Create a dictionary that allows rooms to be linked together using directions as keys, enabling the user to travel from one room to another easily
4) The use of property setters to override any mis-inputs of data
5) The use of aggregate and composition to enable the program to pass objects as parameters into different class methods to manipulate specific attributes of the desired object
6) Use class attributes to ensure once the room is visited once the subroutine is not played again to maintain data integrity

5) Login System:

1) Different logins to secure the data saved for a particular user so that multiple users could play on the same system and compare their outcomes
2) Save data to an external file out of the main program to ensure safekeeping once the game has finished
3) Encrypting the password and essential data in the file using hashing methods to enable privacy and security
4) Ensure only users with the right login credentials can access the data

6) User Menu:

1) The user should be provided with a multitude of options to choose from at the end of each subroutine/ room
2) These options should use complex user defined Object-oriented programming models such as composition and aggregation in order to enable the user to directly interact with character and room inventories
3) The appending and removal of items from user and character inventories should be implemented using complex list operations storing objects
4) The user should get a limited number of tries to access the menu and should be able to exit it whenever needed

7) Save/Load Menu:

1) Menu to navigate different levels and different difficulties of the story mode:
   Upon finishing each level or era of history the user would be able to play that level again being able to overwrite their current data and change their decisions affecting the finality of the game. Each decision would be stored in a database and the user would be able to overwrite that data with a simple click of a button. This would be done using the file handling methods that come built in python such as open. The overwriting of data could be implemented using stack operations to return to latest saved stage of the game
2) Graph nodes and tree traversal should be used to notify the branch of the decision tree the user has reached in the program
3) Erase data from a user profile to start a new game
4) Check all the easter eggs/ artefacts they have collected:
   A list of all the artefacts collected by the user and stored on a database would be listed
5) Have an option to pause the game and decide to save the data, reset from their last checkpoint, reset the level or quit to main menu
6) Use a merge sort or a bubble sort algorithm to determine the exact point of entry to load the user into the game with the previously input data intact

Problems and Limitations of the Game

1. Repeated Situations in Game - Once player finishes the game and starts over. Despite the objective and the motive of the game, which is to avoid linearity, if a user is to play the game more than twice, they would start to run into the same kind of situations and exhaust the options more quickly
2. Privacy of Data Stored on JSON file
3. Creating a difficult boss battle for the final stage that provides a bit of challenge to the user but isn't controlled by the user but by the program itself
4. Displaying, storing and keeping track of so many different characters and rooms to make sure when the user induces a change in an attribute that change is reflected throughout the whole program for data integrity
5. Creating a save and load system that runs in parallel with the Register and Login system - Creating a system that allows the user to save whatever data (i.e., all decisions) they have entered at any point in the game and reload them back into the exact point of the game they left at. All this data would be stored on an external file and would be retracted whenever needed
6. A problem commonly encountered with text-based adventure games is the ability to keep the player engaged, this often difficult without 2D graphics or combat systems, as such even old solutions to this problem relied on creating multiple puzzles and complex maps to keep the user engaged

Solutions:

A solution to the Privacy problem of storing data on JSON files would be to encrypt the data using hashing algorithms, this not only makes the data unreadable but there is no way to unhash the data so it remains safe within the contents of the file.

The solution for creating a complex AI/algorithm for the final maze boss battle could be using a recursive algorithm that uses either breadth first, depth first or random traversal methods to scour the maze acting as artificial intelligence for the user to compete with

A solution to keeping track of different characters and maps is to create multiple classes and use complex user defined Object-Oriented Programming models such as inheritance, compositions and aggregation to keep track of multiple objects while still accessing their methods and attributes without giving up data integrity

A solution to the repetition of situations is making sure every decision input by the player affects their attributes and thus in turn these attributes affect the level of difficulty of their minigames, as well as the number of choices they have. E.g. having a certain threshold of charisma before stealing from another character. Another solution is implementing different designs within the minigames such as having 3 different traversal methods for the maze or having different ball speeds for the break out game

A solution to maintain player engagement to optimum level is to create several minigames defined as huge, complex subroutines using the library of pygame. Often mathematical models are required to help simulate the minigames, such as keeping time in the background or modelling the collision system in the brick game. These models can be represented using a flowchart and are highlighted in the code using comments

Programming Language used:

Python:

Looking at the scope of the problems and limitations I decided to use Python as my programming language due its several inbuilt functions.
Not only python is less tedious but it is also much easier to debug and has many more resources available to it such as finding an esoteric module called the Caesar cipher that allows you to easily encrypt and decrypt data by simply passing in the message and the offset to be used

Examples of libraries that could be used to aid my solutions:

Os Library: This library could be used to help me make the game look better and to clear the screen when required as to remove data from user vision and ask for their input or display a real time timer that updates the user every second

Re Library: This library could be used to help me validate entries using data patterns and the concept of regular expressions to make sure inappropriate data isn't used

Json Library: This library could be used to help me parse data from python to a Json file. This not only allows me to overwrite and update the Json file but allows me to read from it too, enabling me to display user details

Pygame Library: This library is the most important library to help me make minigames that tackles one of my user objectives and limitations. The library aids in the rendering of images and sprites made by the game as well as takes the user input in real time

Base Mechanics and Layout of the Game

Text Based games are usually played in an orderly fashion of inputting text in a manner which allows you to travel to the next point in the game. The travel between rooms for example could be done an infinite number of times but would not help progress the story thus using class attributes I implemented a system which only restricts the user to engage with story of the room once and forces them to progress onto other rooms until the end.
This mind map helps me visualise what the overall structure of the game would look like.

Documented Design

Mechanics:

Each NPC (Non - Player Character) would be initialised at the start of the game with their attributes set to default. The interaction between the user and the NPC would be recorded and the various types of interactions are:

- A series of questions contesting the user's memory and attention paid to the game
- A series of challenges of mini games respective to the storyline they are playing
- Allowing you access to different parts of the game

As a result, many different systems have to be implemented to ensure the smooth running of the user interface. This is mainly done using abstraction as the intricacies of complex user defined object-oriented programming methods are kept hidden from the user. Most subroutines usually need singular inputs from the user or run on their own using recursive algorithms or random modules

The user's interaction with the game is not only limited to the manipulation of the story line and its characters but also the setting they are based in.

The sections below are split into the main systems of design ranging from class implementations, file opening implementations to implementation of recursive algorithms. Each section is provided with an image in the form of either a flowchart or data flow diagram that helps explain the process

Login System:

As described in the solution to the problem of keeping track of user data as well as privacy, a login system is to be implemented; using the guidance of this data flow diagram below, a system using the Json module can be implemented in python. The Register subroutine validates the password and the username and then adds them to the file calling the 'AddUserinfo' which uses the load and dump methods in the Json module

The login subroutine prints out the current details of the user by matching the username per line and then displaying the contents

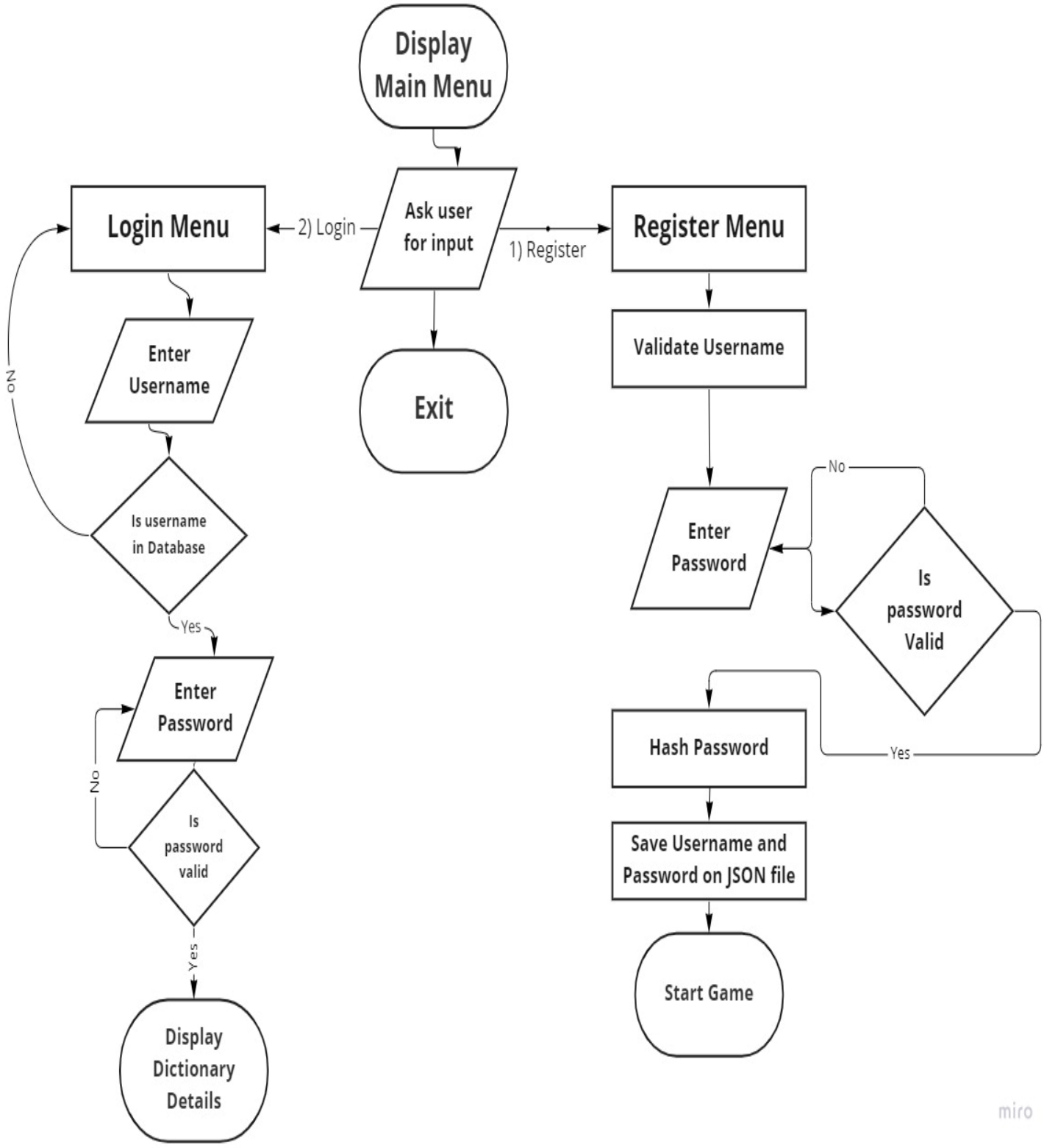
The AddUserinfo and check for user subroutines using python's inbuilt file handling systems with open and close to write and read from lines

The login system also uses the hashlib module to encrypt the passwords by hashing it and then comparing the hashed password stored on the file with the password input during login to validate.

Updating the fields and the keys after major decisions are made is an important feature that is implemented using 2 different subroutines, 1 for a specific field and 1 for the whole line itself

A problem that occurred regularly with this implementation of design is often the file was opened in 'append mode' instead of 'write mode' as I did not want to overwrite the data in the file instead, I wanted to update it. The aforementioned append mode however started at the last line of the file and left the very first line empty. In order to tackle this problem, I used a simple if loop to check if the first line was empty using 'file.readlines()' and continued on even if it was. As a fail-safe to future errors like these i decide to catch a specific type of error that occurs when appending irregular data to a file, the error being: `JSONDecodeError`

This error stopped the file from being updated thus I caught it using exception handling methods (Try and Except) and forced the program to continue to the other line. Thus, it's important if trying to implement my design in a language of your choice that the empty line is caught and does not interfere with the program


Display Main Menu
Ask user for input
2) Login
1) Register
Exit
Login Menu
Enter Username
No
Is username in Database
Yes
Enter Password
No
Is password valid
Yes
Display Dictionary Details
Register Menu
Validate Username
Enter Password
No
Is password Valid
Yes
Hash Password
Save Username and Password on JSON file
Start Game
miro

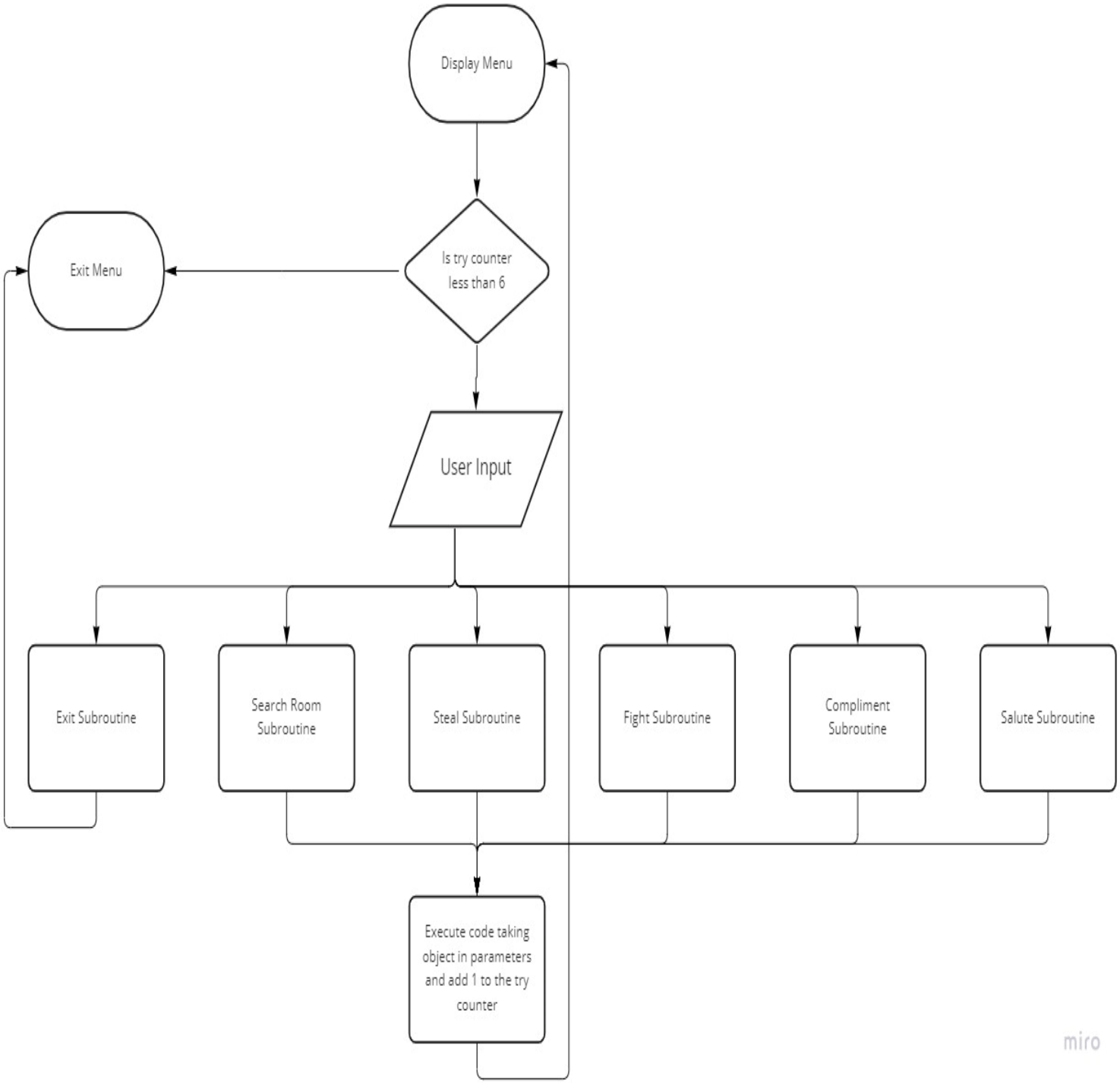
Menu Mechanics:

Using association for every option in the menu, a solution to access objects of multiple classes is to pass those objects as parameters for the menu subroutine, which can be subjective to each room and each character.

A problem I encountered was appending an object, that the user has successfully stolen or searched through the room, to their inventory, as the object itself would be stored but when the user would want to use it, it would be syntactically incorrect to ask for a string input and compare it to an object

A solution to this problem was appending the object to the inventory but then comparing the input to the object's name attribute. If the object's name was identical to the input given then the method could be carried out. Another similar problem of comparing strings and objects was printing out the user's inventory at the end of every room for the current user stats method
I used list operations to each items name onto another list and then displayed the list

A caution to this method is losing track of changes done to the code as for example appending an object that is declared as a null type ('None') will contain the error: Attribute Error: 'None Type' object has no attribute 'self'. And this will cause the program itself to crash


Display Menu
Is try counter less than 6
Exit Menu
User Input
Exit Subroutine
Search Room Subroutine
Steal Subroutine
Fight Subroutine
Compliment Subroutine
Salute Subroutine
Execute code taking object in parameters and add 1 to the try counter
miro

Map Creation:

As mentioned in the Room class design page, the map layout of the game was created using the diagram below and implemented using classes and dictionaries.
A major important feature of all the rooms is the ability to record the user's first ever interaction with the room

Story Line:

The story line is manipulated according to the users' decisions however despite the promise of the game includes some linearity limitations to it as to finish the game no matter what you do, you will reach the final boss battle with Hitler
The games story progression is entirely dependent on the user to take the subtle hints and clues as to what to do next
The game could be played in a very passive way where the user could skip every menu system

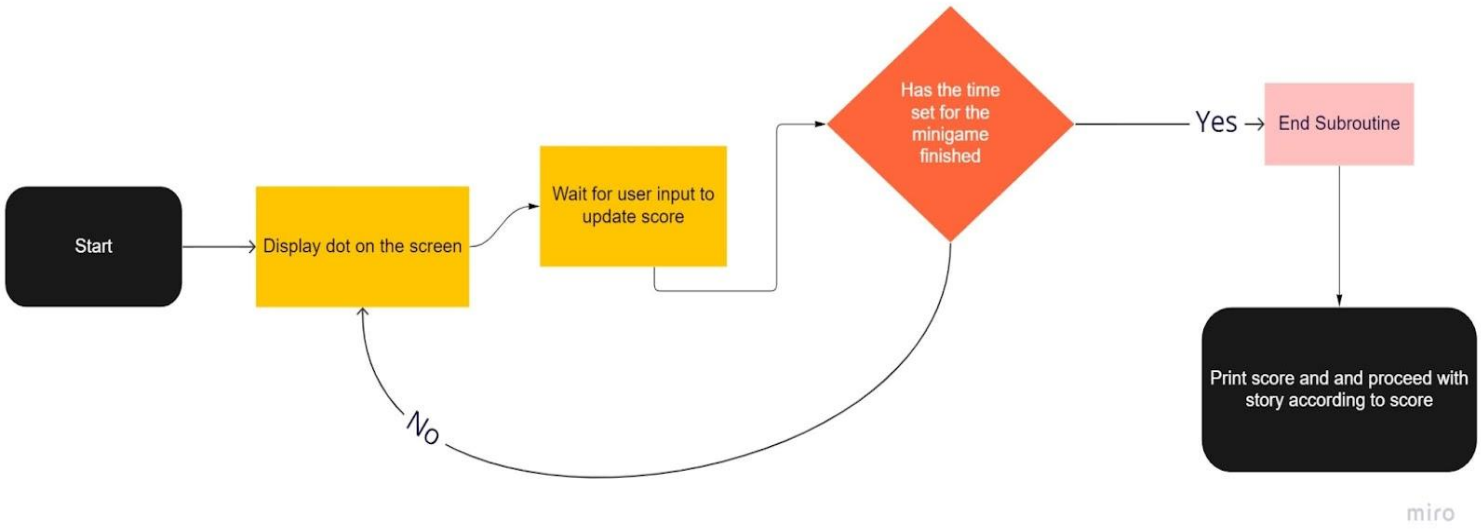
Minigames:

The design and choice of the minigames could be subjective to change for each language. The main choice of minigames chosen was due to the essential role of the module pygame. The pygame module help easily blit images and text onto the screen, but if the programming language of your choice does not have such modules, then simplistic game designs that are engaging serve the purpose too

Minigame at the Shooting Range [Aim Trainer] :

This is the more simplistic design of the minigame, where in reality functions like render and blit tackle the problem of displaying the dots and collision of dots is handled using mathematical models

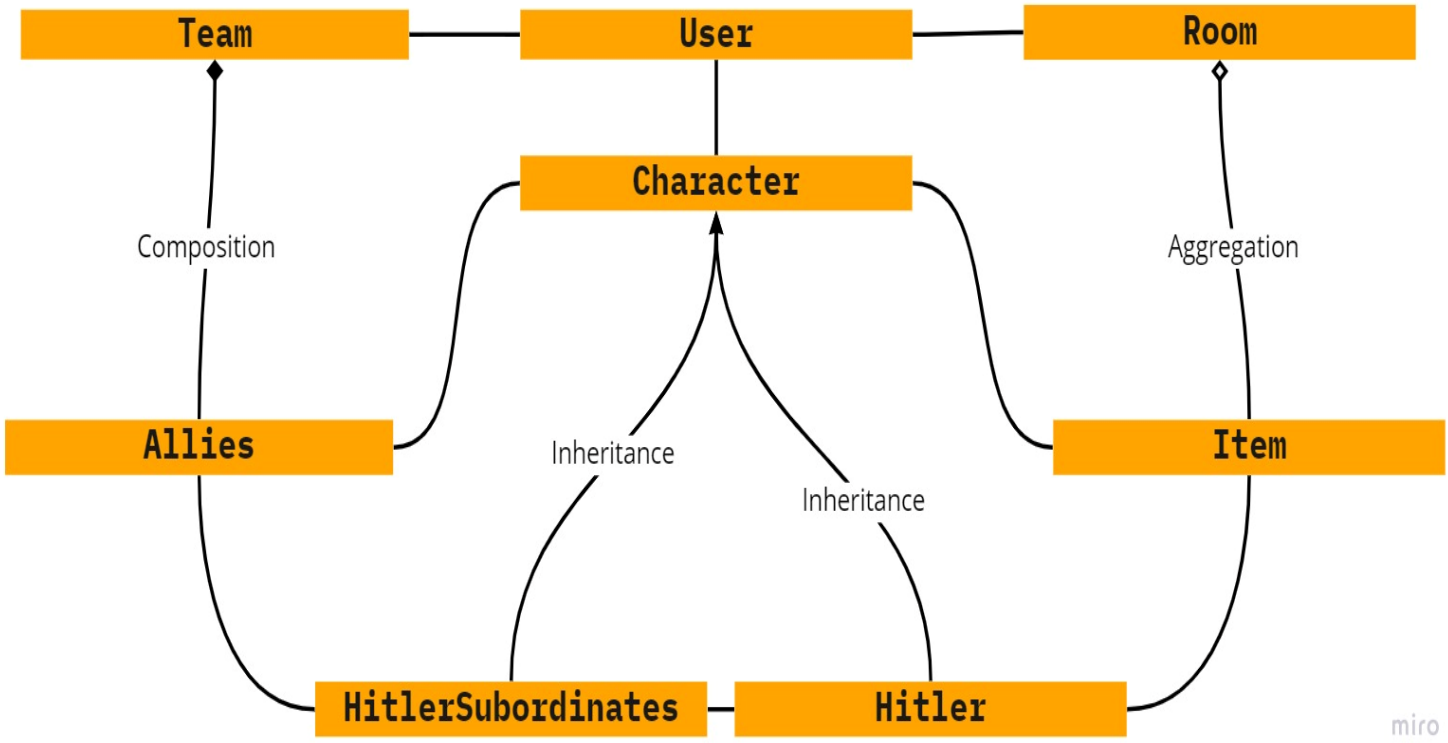
Flowchart Diagram explaining the process:



Classes Design:

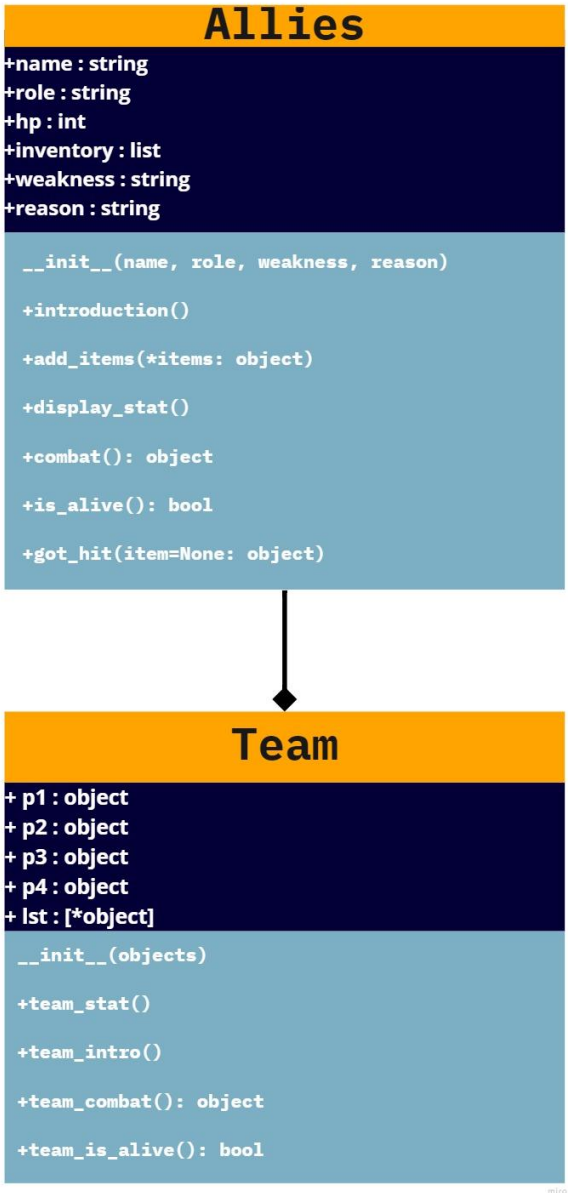
The use of property declarators using the '@' symbol helps establish a more efficient way of coding that maintains data integrity. The only thing common in all the classes is the use of property declarations. This not only makes the code more efficient as in order to set or retrieve attributes I do not need to remember the exact name given to the method.

This is often useful while coding the objects' attributes in different classes as methods aren't auto suggested. Property declarations help keep track of class attributes and pose the same functionality as setters and getters but in a more efficient way.



Composition Association:

The team class uses composition to create objects from the Allies class during instantiation. Each object is stored in attributes such as player 1 object stored in self.p1. The team class could not exist without the Ally objects thus in turn is dependent on the Ally class to be created
For Team class's methods, the team combat would probably be the most complex one as it returns an object that will be used as a parameter to a method in the Hitler class. This returned object is retrieved from the class's attributes (i.e., from the ally object) thus is retrieved from the object's inventory. This method alone involves the use of 4 different classes using association. The item from the item class is stored in the ally class, used in the team class and returned to the Hitler class.
The team class method not only uses the ally's class's objects and attributes but also its methods



Class User:

The class user will have the most methods using property declarations as often most attributes in other classes are pre-set and the user is not allowed to manipulate them.
The class user has more simplistic design implementations in regards to setting methods as most methods serve the purpose either manipulating an attribute or printing or returning an attribute.

The most complex method in this class would probably the use of combat as it uses objects stored in the user's inventory as well as takes in another class's attributes to manipulate and compare.

Character Class (Parent Class) :

The class is the parent class to the 'hitlersubordinates' class and the 'Hitler' class and thus contains the inherited attributes and methods of those classes. This class serves the purpose of providing the basic and blank slate for creating any NPCs (Non player characters) within the game.

These NPCs can then be attributed with additional methods and attributes that are appropriate for their role in the game. This helps keeps consistency and avoids repetition of code using abstraction

User

+name : string
+charisma : int
+passion : int
+skill : int
-hp : int
+charm : int
+inventory : list
+fame : int

__init__(name, charisma, passion, skill, charm, fame)

@property
-name(): string
@setter
name(value: string): string

@property
charm():int
@setter
charm(value: int): int

@property
charisma(): int
@setter
charisma(value: int): int

@property
skill(): int
@setter
skill(value: int): int

@property
passion(): int
@setter
passion(value: int): int

@property
hp(): int
@setter
hp(value: int): int

@property
fame(): int
@setter
fame(value: int): int

+change_charisma(d: bool): int

+change_passion(d: bool): int

+change_skill(d: bool): int

+change_hp(d: bool): int

+change_inventory(item: object, value: bool): list

+change_charm(d: bool): int

+combat: bool

+is_alive : bool

+display_user_combat_stats

Character

name: string
role: string
description: string
risk: int
weakness : list(string)
fame : int
inventory: list

__init__(name: string, role: string, description: string, risk: int, weakness: string, fame: int)

@property
name(): string
@setter
name(value: string)

@property
role(): string
@setter
role(value: string)

@property
description(): string
@setter
description(value: string)

@property
weakness(): string
@setter
weakness(value: int)

@property
risk(): string
@setter
risk(value: int)

@property
fame(): string
@setter
fame(value: int)

+fight(user_attributes: object)

+steal(user_attributes: object)

+salute(user_attributes: object)

+change_inventory(item: object, value: bool)

Aggregation Association:

The Room class and the Item class are linked together using aggregation association. Where there are items stored in the room, thus stored in the room class's inventory attribute, however, the item can exist even if the room does not exist such as for items that are stored in user object's inventory or character object's inventory.

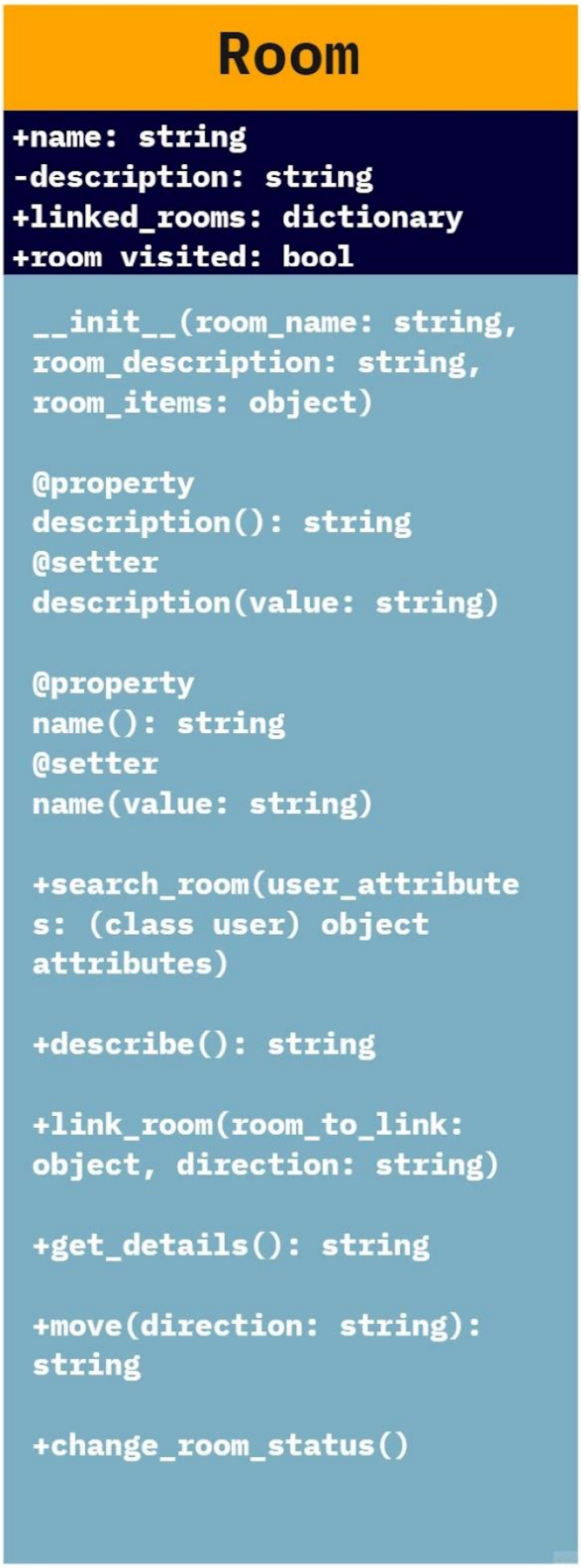
Set item placement is the most important method to this aggregation association relationship as it returns a list of the coordinates where the item is hidden/located in the room. This is done using the random class. This list is then returned to an equally important method in the Room Class called search room, which asks the user for integers to search the room storing their answers in a 2d list till their answer matches the actual location of the item object. This is done using advanced 2d list operations and splicing.

The room class also implements the use of dictionaries to store the locations of other rooms in relation to one room. This is implemented using the direction as the key and the room object in that direction as the value itself. This implementation in combination with the describe method helps create a very reliable, efficient and simplistic structure which could be used in the room transition method later in the program, during the phase of map creation

The actual move method is what describes the new location of the user in context of the room and handles the validation of the direction. Using abstraction, I did not need to implement a system to catch invalid inputs when asked for directions on where to go as this is self-implemented in the move method and the room transition subroutine could be focused on linking different Room subroutines.

Finally, the room status, although simple looking, is of great essence to the smooth running of the program to make sure that when a room is first visited by a user it is recorded in order to avoid repetition of rerunning the same routines. As the attribute deals in Boolean values, once the room is visited the room visited status is changed to True and if the user decides to visit the room again, they are redirected to another room not visited. The purpose of this entire feature is to maintain data integrity and data abstraction from the user


Item
+name: string
-description: string
+item_placement: list
__init__(name: int):
@property
name(): string
@setter
name(value: string)
@property
description(): string
@setter
description(value: string):
+set_item_placement() : list

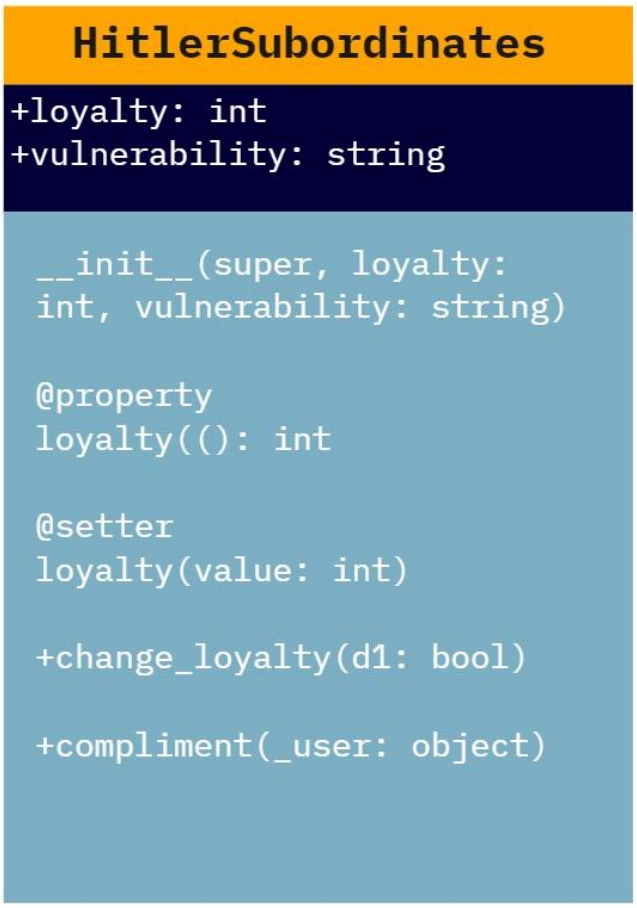

Room
+name: string
-description: string
+linked_rooms: dictionary
+room visited: bool
__init__(room_name: string, room_description: string, room_items: object)
@property
description(): string
@setter
description(value: string)
@property
name(): string
@setter
name(value: string)
+search_room(user_attributes: (class user) object attributes)
+describe(): string
+link_room(room_to_link: object, direction: string)
+get_details(): string
+move(direction: string): string
+change_room_status()

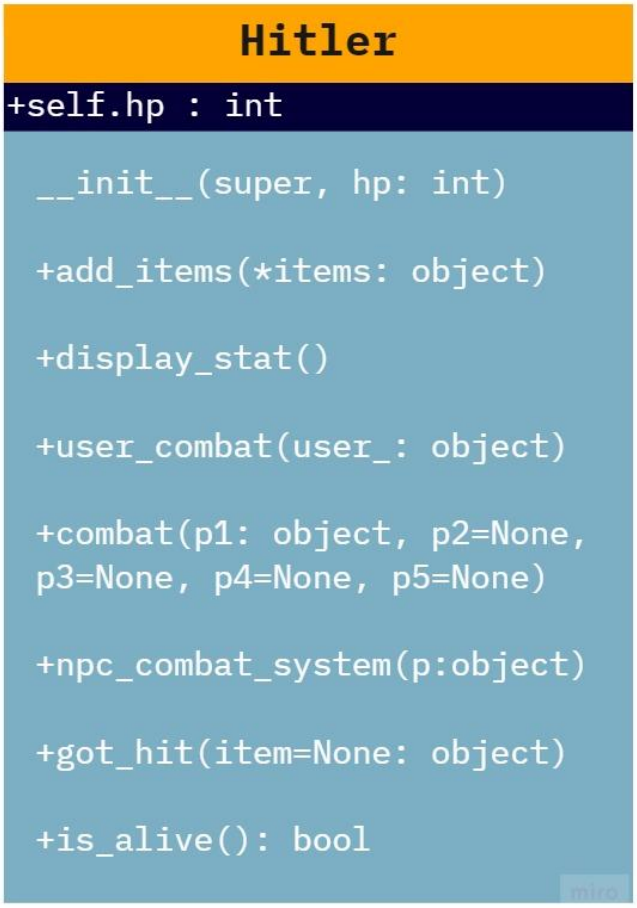
Inheritance:

These 2 classes below are the examples of implementing inheritance in my program. Both of them inherit from the Character class as shown in the constructor function; super.init()
Thus, they contain all the attributes and methods of the Character class. However, I plan to implement a different set of characters for different situations in the game.

Such as a sub character class called Hitler's Subordinates for when you play as a soldier in the Nazi regime. That is why the class Hitlersubordinates has the additional attributes of loyalty and vulnerability, because as a soldier you might be able to dissuade them onto your side. On the contrary, for the situation where the user plays as a prisoner in one of the camps, there would be no use for such attributes or class methods that manipulate these attributes and camp officers would have attributes of their own such as 'harshness'.

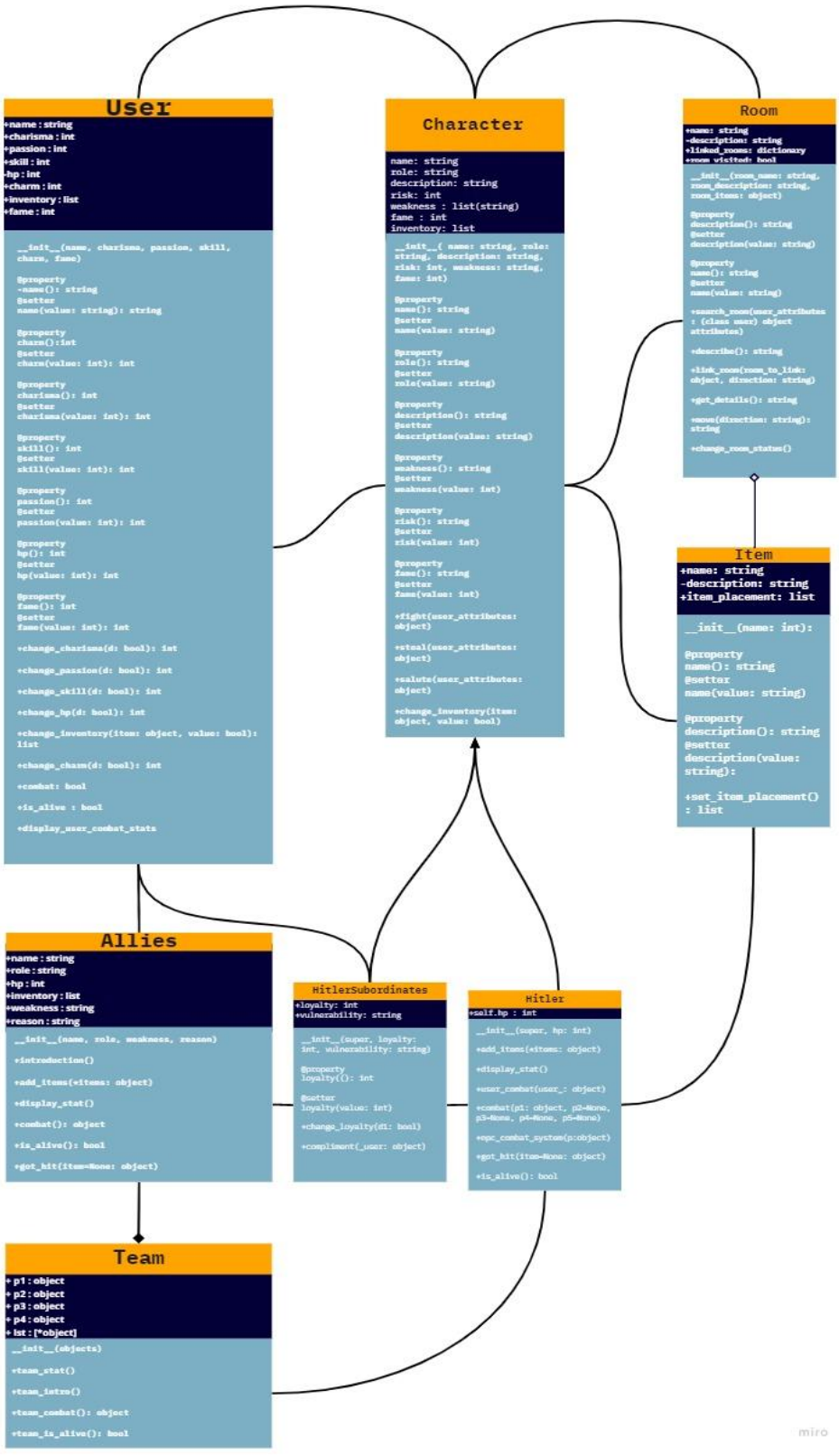
Even the class of Hitler would be not executed if the user chooses to play the scenario where they help Napoleon Bonaparte escape. However, in that scenario I would could have probably created a class called 'Final Boss' and made all final boss decisions inherit from that class, this could be an alternative design to my Boss Battle class system
As shown in the class uml diagrams below, all the parameters that are passed into methods for the class 'Hitler' are objects. This defines the use of association to use object attributes from other classes to be manipulated or compared to in the current class





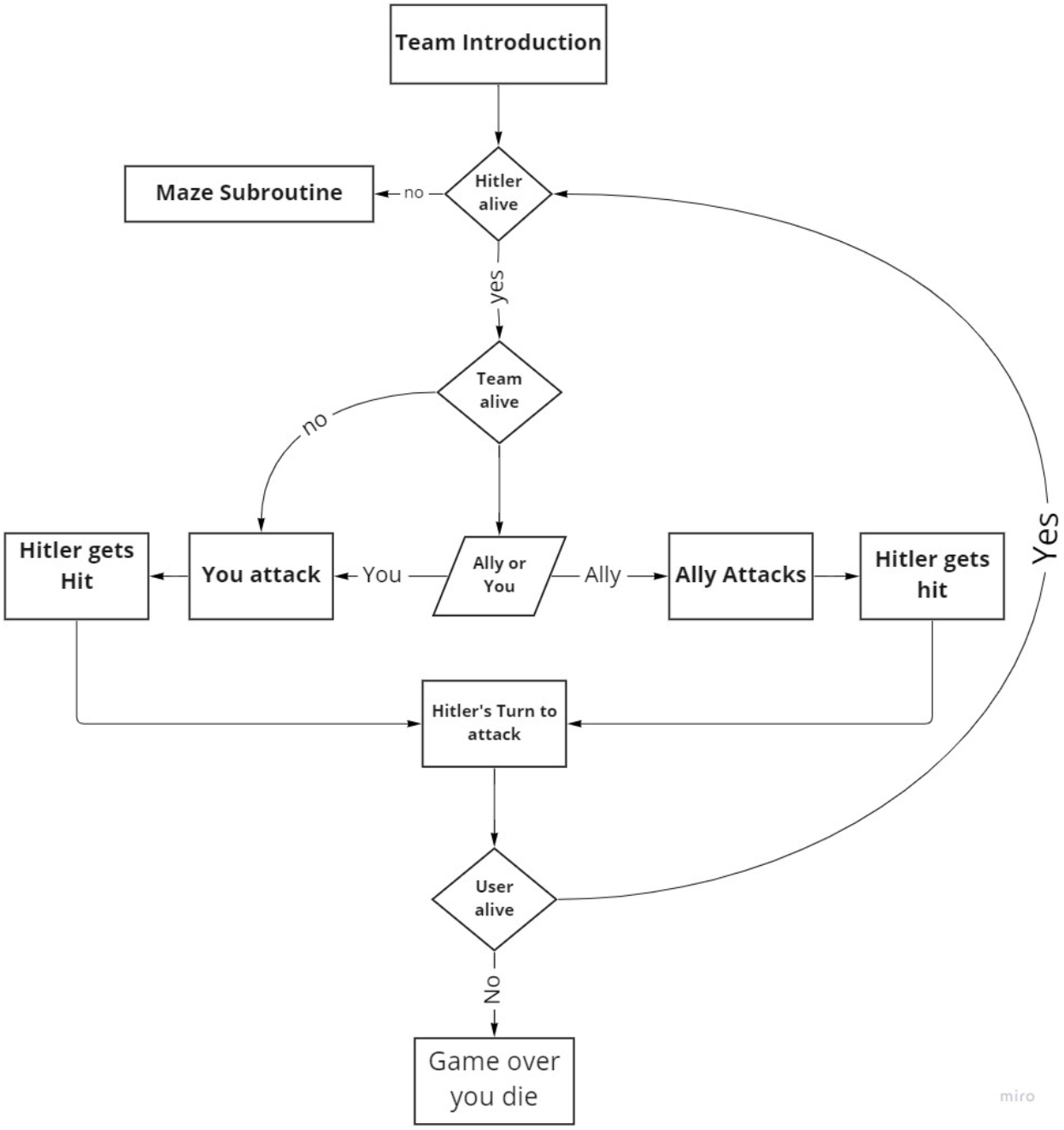
Linking Structure:

In conclusion, using complex user defined object-oriented programming techniques such as association, aggregation, composition and inheritance, helps me handle user inputs and data manipulation in a more efficient way to present a more methodical approach to my game.



Combat System:

The classes created: Allies, Hitler and Team are all set up for the final boss battle to take place. As you can see here each vital decision displayed by a diamond is an actual method in either of the classes that returns a Boolean value. While the input sets up a stage for either the user to attack or their ally, each attack returns an object to pass into ‘Hitler gets hit’ which manipulates his hp.



Final Maze Algorithm:

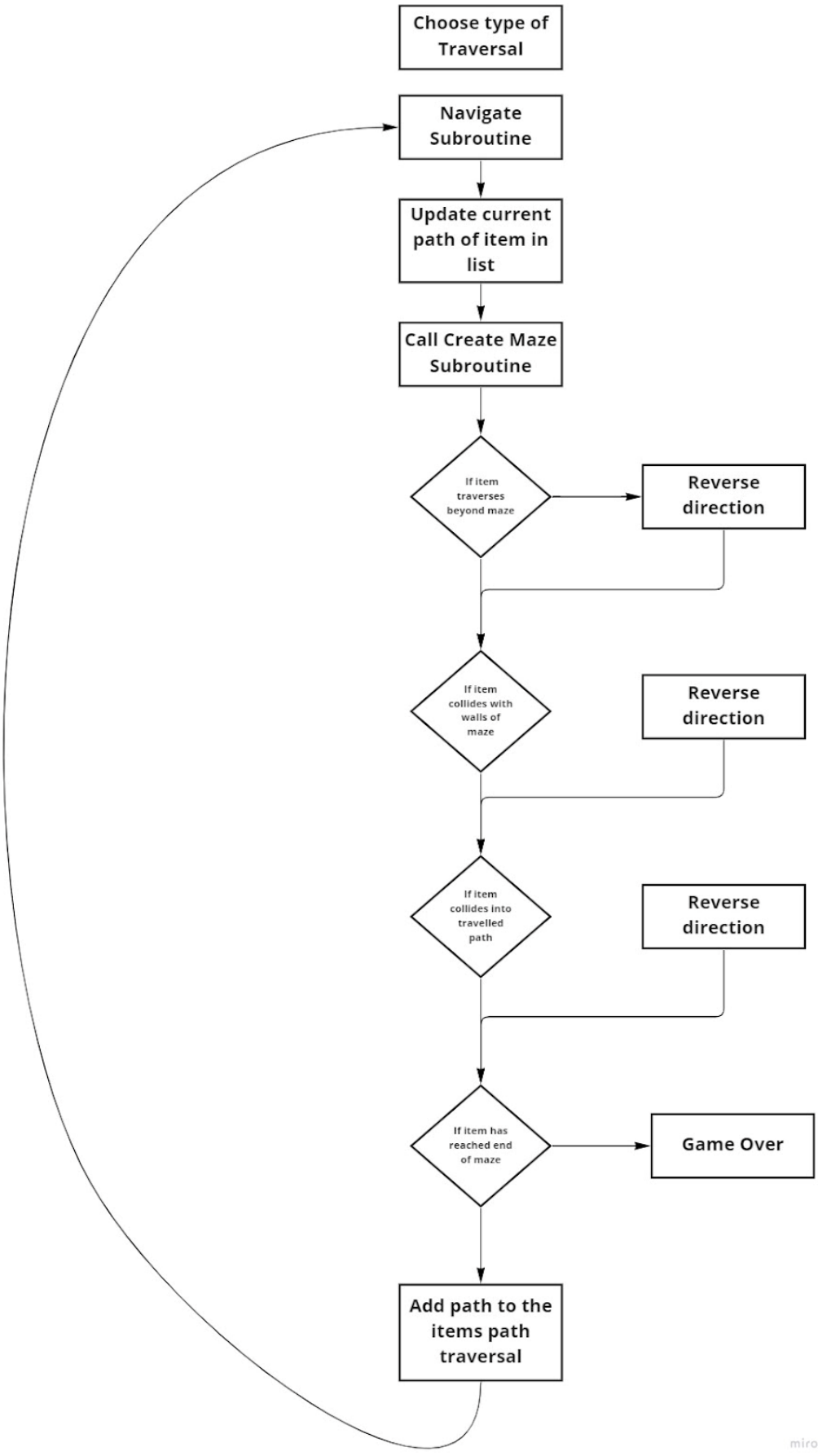
As shown in the data flow diagram below the only base condition to exit the loop is if the item reaches the end of the maze, until then the algorithm calls upon itself with a new path every cycle. As shown in the diagram below there should be a way to reverse or avoid going into restricted areas of the maze, in python the selection is built in such a way that allows me to just use ‘continue’ statements.

For any other language the basic principle is to ensure that the item is only added to the maze path if it overcomes all these conditions.

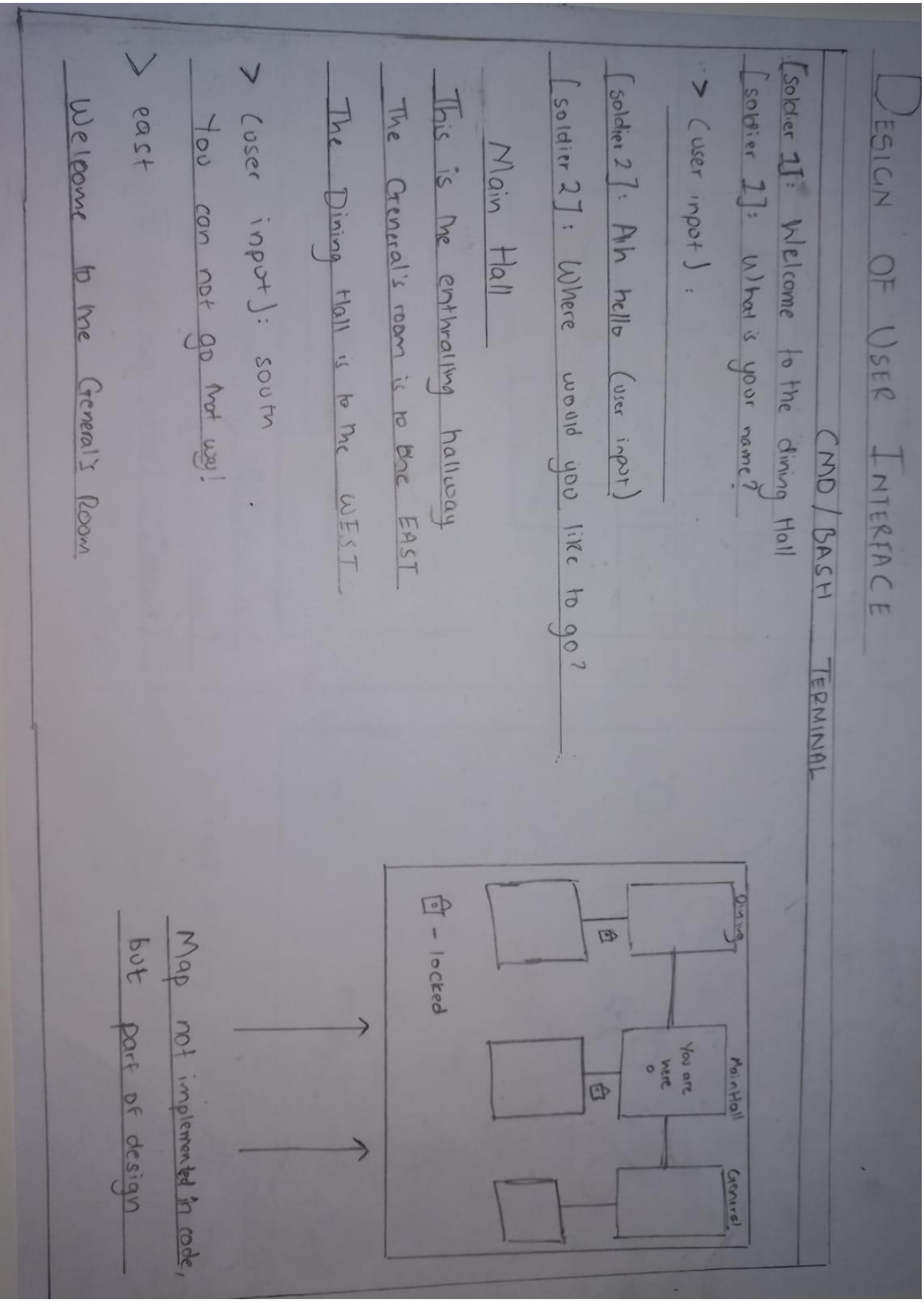
The algorithm in code is set up in such a way where:

- The maze is retrieved from a csv file containing values and converted into a list
- The maze is then drawn using pygame mechanics
- The traversal method is chosen
- The restriction conditions are compared and are passed through
- If all the conditions are met the item is added to the maze path
- The draw maze function is called again and renders and blit(s) the images onto the screen according to the items in the list
- If the end of the maze has not reached the whole cycle starts again

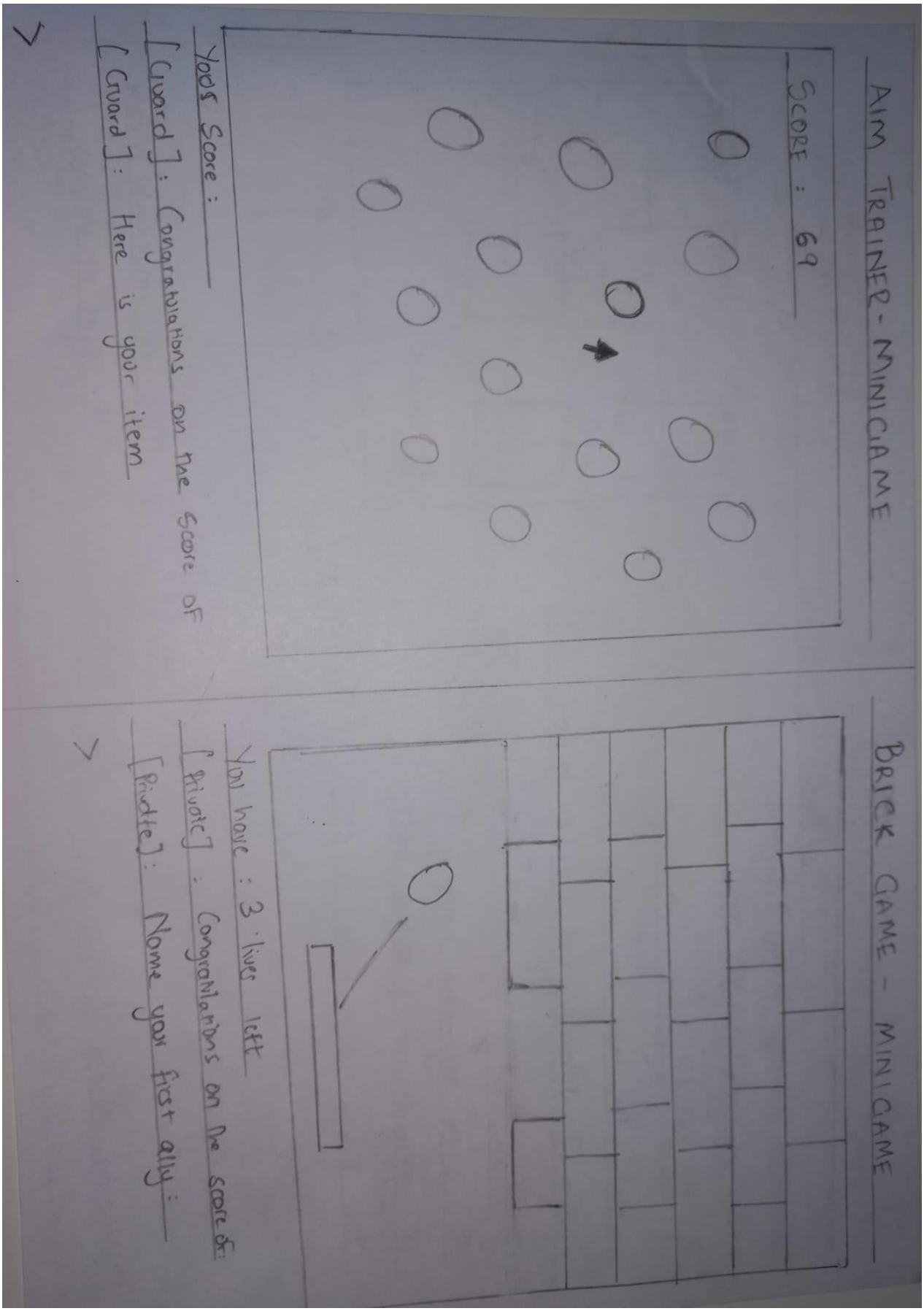
The important thing to note here is that the actual adding of the items and replacement of the position of the item in the maze is done in the 2d list. However, each item from that 2d list is then given an image which blit(s) onto the screen. This could be compared to refresh rates on the screen updating the game information on the screen with each refresh


Choose type of Traversal
Navigate Subroutine
Update current path of item in list
Call Create Maze Subroutine
If item traverses beyond maze
Reverse direction
If item collides with walls of maze
Reverse direction
If item collides into travelled path
Reverse direction
If item has reached end of maze
Game Over
Add path to the items path traversal
miro

What the game should look like on paper (terminal):



Minigame Design on Paper:



Laksh Agarwal Center Number: 74636 Candidate Number: 1104

Technical Solution

```
from random import randint


class user:
    def __init__(self, name, charisma, passion, skill, charm, fame):
        self.name = name
        self.charisma = charisma
        self.passion = passion
        self.skill = skill
        self.hp = 10
        self.charm = charm
        self.inventory = []
        self.fame = fame

    # property declarations that act as setters and getters

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def charm(self):
        return self._charm

    @charm.setter
    def charm(self, value):
        self._charm = value

    @property
    def charisma(self):
        return self._charisma

    @charisma.setter
    def charisma(self, value):
        self._charisma = value
```

```
40
41      @property
42      def skill(self):
43          return self._skill
44
45      @skill.setter
46      def skill(self, value):
47          self._skill = value
48
49      @property
50      def passion(self):
51          return self._passion
52
53      @passion.setter
54      def passion(self, value):
55          self._passion = value
56
57      @property
58      def hp(self):
59          return self._hp
60
61      @hp.setter
62      def hp(self, value):
63          self._hp = value
64
65      @property
66      def fame(self):
67          return self._fame
68
69      @fame.setter
70      def fame(self, value):
71          self._fame = value
72
73      def change_charisma(self, d):
74          if d:
75              self.charisma += 1
76          else:
77              self.charisma -= 1
78
79      def change_passion(self, d):
80          if d:
```

```
 81              self.passion += 1
 82          else:
 83              self.passion -= 1
 84
 85      def change_skill(self, d):
 86          if d:
 87              self.skill += 1
 88          else:
 89              self.skill -= 1
 90
 91      def change_hp(self, d):
 92          if d:
 93              self.hp += 1
 94          else:
 95              self.hp -= 1
 96
 97      def change_inventory(self, item, value):
 98          if value:
 99              self.inventory.append(item)
100              print('You have successfully acquired',
    item.name)
101          else:
102              self.inventory.remove(item)
103
104      def change_charm(self, d):
105          if d:
106              self.charm += 1
107          else:
108              self.charm -= 1
109
110      def change_fame(self, d):
111          if d:
112              self.fame += 1
113          else:
114              self.fame -= 1
115
116      def combat(self):
117          self.display_user_combat_stats()
118          if len(self.inventory) != 0:
119              while True:
120                  d = input('Would item would you like
```

```
 to fight with:\n>').lower().strip()
                for i in self.inventory:
                    if d == i.name:
                        print(self.name, 'decides to use', i.name)
                        self.inventory.remove(i)
                        return i
                    else:
                        print(self.name, 'does not have this item')
                        break
        else:
            print('You do not have any item to use so you use a basic attack')
            return None

    def is_alive(self):
        if self.hp > 0:
            return True
        else:
            return False

    def display_user_combat_stats(self):
        print('Your remaining hp is:', self.hp)
        lst = []
        for i in self.inventory:
            lst.append(i.name)
        print('You have:', lst, 'in your inventory')


class Item:
    def __init__(self, name):
        self.name = name
        self.description = None
        self.item_placement = []

    # property declarations that act as setters and getters

    @property
    def name(self):
```

```
157         return self._name
158 
159     @name.setter
160     def name(self, value):
161         self._name = value
162 
163     @property
164     def description(self):
165         return self._description
166 
167     @description.setter
168     def description(self, value):
169         self._description = value
170 
171     # Uses a 2d list to set the item in a location
    in the room, the location is then returned to class
    room
172 
173     def set_item_placement(self):
174         ls = [[0, 0, None], [0, 1, None], [0, 2,
    None], [1, 0, None], [1, 1, None], [1, 2, None]]
175         ls[randint(0, len(ls) - 1)][2] = self.name
176 
177         for x in range(len(ls)):
178             if ls[x][2] == self.name:
179                 self.item_placement.append(ls[x][0])
180                 self.item_placement.append(ls[x][1])
181             else:
182                 continue
183 
184         return self.item_placement
185 
186 
187 class Characters:
188     def __init__(self, name, role, description, risk
    , weakness, fame):
189         self.name = name
190         self.role = role
191         self.description = description
192         self.risk = risk
193         self.weakness = [weakness]
```

```
194         self.fame = fame
195         self.inventory = []
196 
197     # property declarations that act as setters and
    getters
198 
199     @property
200     def name(self):
201         return self._name
202 
203     @name.setter
204     def name(self, value):
205         self._name = value
206 
207     @property
208     def role(self):
209         return self._role
210 
211     @role.setter
212     def role(self, value):
213         self._role = value
214 
215     @property
216     def description(self):
217         return self._description
218 
219     @description.setter
220     def description(self, value):
221         self._description = value
222 
223     @property
224     def weakness(self):
225         return self._weakness
226 
227     @weakness.setter
228     def weakness(self, value):
229         self._weakness = value
230 
231     @property
232     def risk(self):
233         return self._risk
```

```
234
235     @risk.setter
236     def risk(self, value):
237         self._risk = value
238
239     @property
240     def fame(self):
241         return self._fame
242
243     @fame.setter
244     def fame(self, value):
245         self._fame = value
246
247     # Take in user object as parameter to implement
    association in the form of calling the objects
    methods as they are
248     # public
249
250     def fight(self, user_attribute):
251         temp = None
252         combat_item = input("What do u decide to
    fight with\n>").lower().strip()
253         for i in user_attribute.inventory:
254             if i.name == combat_item:
255                 temp = i.name
256
257         if combat_item == temp:
258             if combat_item in self.weakness:
259                 print(self.name, 'apologizes for
    picking a fight with you')
260                 print('you defeat', self.name, 'but
    you lose your item too')
261                 user_attribute.change_fame(True)
262                 for i in user_attribute.inventory:
263                     temp = i.name
264                     if temp == combat_item:
265                         user_attribute.
    change_inventory(i, False)
266                         break
267             else:
268                 print('You lost to', self.name, '
```

```
your reputation has been tarnished')
                user_attribute.change_fame(True)
        else:
            print('Sorry you do not own this item')

    def steal(self, user_attribute):
        temp = None
        command = input('What would you like to
steal\n> ').lower().strip()
        for i in self.inventory:
            temp = i.name
        if temp == command:
            if user_attribute.charisma >= self.risk:
                print('you successfully have stolen'
, command, 'from', self.name)
                for i in self.inventory:
                    temp = i.name
                    if temp == command:
                        user_attribute.
change_inventory(i, True)
                        self.change_inventory(i,
False)
                        break
            else:
                print('you are caught red handed and
 now your reputation is tarnished')
                user_attribute.change_fame(False)
        else:
            print('This item does not exist in',
self.name, "'s inventory")

    def salute(self, user_attribute):
        if user_attribute.fame < self.fame:
            print('you salute', self.name)
            print('you gain their respect')
            user_attribute.change_charisma(True)
        else:
            print('[', self.name, ']', ':', 'oh
thank you sir for saluting me')
            user_attribute.change_charisma(True)
            user_attribute.change_fame(False)
```

```
302
303     def change_inventory(self, item, value):
304         if value:
305             self.inventory.append(item)
306         else:
307             self.inventory.remove(item)
308
309
310 # Example of Inheritance
311
312 class HitlerSubordinates(Characters):
313     def __init__(self, name, role, description, risk
    , weakness, fame, loyalty, vulnerability):
314         # Super function used to inherit the
    attributes and methods of the Characters class
315         super().__init__(name, role, description,
    risk, weakness, fame)
316         self.loyalty = loyalty
317         self.vulnerability = vulnerability
318
319     @property
320     def loyalty(self):
321         return self._loyalty
322
323     @loyalty.setter
324     def loyalty(self, value):
325         self._loyalty = value
326
327     def change_loyalty(self, d1):
328         if d1:
329             self.loyalty += 1
330         else:
331             self.loyalty -= 1
332
333     # Take in user object as parameter to implement
    association in the form of calling the objects
    methods as they are
334     # public
335
336     def compliment(self, _user):
337         compliment = (input('your compliment: ')).
```

```
337 split()
338         if self.vulnerability in compliment:
339             print('you successfully charm', self.
    name)
340             _user.change_fame(True)
341         else:
342             print('how dare you to try to sweet talk
     me')
343             _user.change_fame(True)
344 
345 
346 class Hitler(Characters):
347     def __init__(self, name, role, description, risk
    , weakness, fame, hp):
348         super().__init__(name, role, description,
    risk, weakness, fame)
349         self.hp = hp
350 
351     def add_items(self, *items):
352         for item in items:
353             self.inventory.append(item)
354 
355     def display_stat(self):
356         print(self.name, 'has:\n')
357         lst = []
358         for i in self.inventory:
359             temp = i.name
360             lst.append(temp)
361         print(lst)
362         print(self.name, 'has', self.hp, 'hp
    remaining')
363 
364     # Take in user object as parameter to implement
    association in the form of calling the objects
    methods as they are
365     # public
366 
367     def user_combat(self, user_):
368         print("It is Hitler's turn to attack")
369         self.display_stat()
370         print('Hitler decides to attack you')
```

```
371         if len(self.inventory) != 0:
372             item_choice = randint(0, (len(self.
    inventory) - 1))
373             use = self.inventory[item_choice]
374             print('Hitler uses', use.name, 'on you')
375             self.inventory.remove(use)
376             print('It is super effective')
377             user_.hp -= 5
378         else:
379             print('Hitler uses a basic attack on you
    ')
380             user_.hp -= 3
381 
382     def combat(self, p1, p2=None, p3=None, p4=None,
    p5=None):
383         choice = randint(1, 5)
384         if choice == 1:
385             self.user_combat(p1)
386         elif choice == 2:
387             if p2 is not None:
388                 if p2.is_alive():
389                     print("It is Hitler's turn to
    attack")
390                     print('Hitler decides to fight'
    , p2.name)
391                     self.npc_combat_system(p2)
392                 else:
393                     self.user_combat(p1)
394             else:
395                 self.user_combat(p1)
396         elif choice == 3:
397             if p3 is not None:
398                 if p3.is_alive():
399                     print("It is Hitler's turn to
    attack")
400                     print('Hitler decides to fight'
    , p3.name)
401                     self.npc_combat_system(p3)
402                 else:
403                     self.combat(p1, p2)
404             else:
```

```
405                 self.combat(p1, p2)
406         elif choice == 4:
407             if p4 is not None:
408                 if p4.is_alive():
409                     print("It is Hitler's turn to
    attack")
410                     print('Hitler decides to fight'
    , p4.name)
411                     self.npc_combat_system(p4)
412                 else:
413                     self.combat(p1, p2, p3)
414             else:
415                 self.combat(p1, p2, p3)
416         elif choice == 5:
417             if p5 is not None:
418                 if p5.is_alive():
419                     print("It is Hitler's turn to
    attack")
420                     print('Hitler decides to fight'
    , p5.name)
421                     self.npc_combat_system(p5)
422                 else:
423                     self.combat(p1, p2, p3, p4)
424             else:
425                 self.combat(p1, p2, p3, p4)
426 
427     def npc_combat_system(self, p):
428         self.display_stat()
429         if len(self.inventory) != 0:
430             item_choice = randint(0, (len(self.
    inventory) - 1))
431             use = self.inventory[item_choice]
432             print('Hitler uses', use.name, 'on', p.
    name)
433             self.inventory.remove(use)
434             if use.name == p.weakness:
435                 print('The attack is highly ,
    immediately killing', p.name)
436                 p.hp -= 10
437             else:
438                 print('The attack is effective')
```

```
439                 p.hp -= 7
440 
441         else:
442             print('Hitler uses a basic attack on', p
    .name)
443             p.hp -= 5
444 
445     def got_hit(self, item=None):
446         if item is not None:
447             print(self.name, 'got hit with', item.
    name)
448             if self.hp > 0:
449                 if item.name == self.weakness:
450                     self.hp -= 5
451                 else:
452                     self.hp -= 3
453             else:
454                 print(self.name, 'has died')
455             if self.hp < 0:
456                 print(self.name, 'has died')
457         else:
458             print(self.name, 'got hit')
459             self.hp -= 1
460 
461     def is_alive(self):
462         if self.hp > 0:
463             return True
464         else:
465             return False
466 
467 
468 class Room:
469     def __init__(self, room_name, room_description,
    room_items):
470         self.name = room_name
471         self.description = room_description
472         self.linked_rooms = {}
473         self.room_visited = False
474         self.room_items = []
475         self.room_items.append(room_items)
476 
```

```
477     @property
478     def description(self):
479         return self._description
480 
481     @description.setter
482     def description(self, value):
483         self._description = value
484 
485     @property
486     def name(self):
487         return self._name
488 
489     @name.setter
490     def name(self, value):
491         self._name = value
492 
493     def search_room(self, user_attributes):
494 
495         # Using aggregation association to assign an
     item to a room object
496 
497         if self.room_items[0] is not '':
498             room_item = self.room_items[0]
499             _user_attributes = user_attributes
500 
501             def search_room_item(_room_item,
    _user_attributes):
502                 print('you have 3 tries to search
    the room using numbers, you cannot search the same
    place '
503                       'again')
504                 lst = []
505                 for i in range(3):
506                     user_input1, user_input2 = None
    , None
507                     while (user_input1 is None and
    user_input2 is None) or ([user_input1, user_input2]
    in lst):
508                         try:
509                             user_input1 = int(input(
    "enter row:"))
```

```
510                                 user_input2 = int(input(
    "enter column:"))
511                             except Exception:
512                                 print('invalid input')
513                         lst.append([user_input1,
    user_input2])
514 
515                 ls = _room_item.set_item_placement()
516 
517                 if ls in lst:
518                     print('You have found',
    _room_item.name)
519                     _user_attributes.
    change_inventory(_room_item, True)
520                     self.room_items.remove(room_item
    )
521                 else:
522                     print('You plundered and
    plundered but could not find anything')
523 
524             search_room_item(room_item,
    user_attributes)
525         else:
526             print('This room does not have any item'
    )
527 
528     def describe(self):
529         print(self.description)
530 
531     # Using dictionaries to store objects and link
    one room to another
532 
533     def link_room(self, room_to_link, direction):
534         self.linked_rooms[direction] = room_to_link
535         # print( self.name + " linked rooms :" +
    repr(self.linked_rooms) )
536 
537     def get_details(self):
538         print(self.name)
539         print("--------------------")
540         print(self.description)
```

```
541         for direction in self.linked_rooms:
542             room = self.linked_rooms[direction]
543             # print(f"{type(room)}, {type(direction
    )}")
544             print("The " + room.name + " is to the "
     + direction)
545 
546     # Method that catches errors and returns the
    room actually moved to
547 
548     def move(self, direction):
549         if direction in self.linked_rooms:
550             return self.linked_rooms[direction]
551         else:
552             print("You can't go that way")
553             return self
554 
555     def change_room_status(self):
556         self.room_visited = True
557 
558 
559 class Allies:
560     def __init__(self, name, role, weakness, reason
    ):
561         self.name = name
562         self.role = role
563         self.hp = 10
564         self.inventory = []
565         self.weakness = weakness
566         self.reason = reason
567 
568     def introduction(self):
569         print(self.name.title(), self.role, 'has
    decided to join your cause as he', self.reason)
570 
571     def add_items(self, *items):
572         for item in items:
573             self.inventory.append(item)
574 
575     def display_stat(self):
576         print(self.name, 'has:\n')
```

```
577         lst = []
578         for i in self.inventory:
579             temp = i.name
580             lst.append(temp)
581         print(lst)
582         print(self.name, 'has', self.hp, 'hp
    remaining')
583 
584     def combat(self):
585         self.display_stat()
586         if len(self.inventory) != 0:
587             while True:
588                 d = input('What item would you like
    your ally to fight with:\n>').lower().strip()
589                 for i in self.inventory:
590                     if d == i.name:
591                         print(self.name, 'decides to
     use', i.name)
592                         self.inventory.remove(i)
593                         return i
594                     else:
595                         continue
596                 print('Your ally does not have this
    item')
597         else:
598             print('Ally does not have any item to
    use so uses a basic attack')
599             return None
600 
601     def is_alive(self):
602         if self.hp > 0:
603             return True
604         else:
605             return False
606 
607     def got_hit(self, item=None):
608         if item is not None:
609             print(self.name, 'got hit with', item.
    name)
610             if self.hp > 0:
611                 if item.name == self.weakness:
```

```
612                     self.hp -= 10
613                 else:
614                     self.hp -= 7
615             else:
616                 print(self.name, 'has died')
617             if self.hp < 0:
618                 print(self.name, 'has died')
619         else:
620             print(self.name, 'got hit')
621             self.hp -= 5
622 
623 
624 class Team:
625     # The use of composition in the constructor to
    set the objects as attributes of the class
626     def __init__(self, p1=None, p2=None, p3=None, p4
    =None):
627         self.p1 = p1
628         self.p2 = p2
629         self.p3 = p3
630         self.p4 = p4
631         self.lst = [self.p1, self.p2, self.p3, self.
    p4]
632 
633     def team_stat(self):
634         total = 0
635         for item in self.lst:
636             if item is not None:
637                 total += int(item.hp)
638             else:
639                 continue
640         print("Your team's total hp is: ", total)
641         for item in self.lst:
642             if item is not None:
643                 _lst = []
644                 for i in item.inventory:
645                     temp = i.name
646                     _lst.append(temp)
647                 print(item.name, 'has:', _lst)
648             else:
649                 continue
```

```
650
651     def team_intro(self):
652         for item in self.lst:
653             if item is not None:
654                 item.introduction()
655
656     def team_combat(self):
657         lst_ = None
658         while True:
659             for item in self.lst:
660                 if item is not None:
661                     if item.is_alive():
662                         print('Your ally:', item.
    name)
663                         lst_ = [item]
664
665             if lst_ != '':
666                 d = input("Who would you like
    fighting from your team\n>").lower().strip()
667                 for i in self.lst:
668                     if i is not None:
669                         if d == i.name:
670                             if i.is_alive():
671                                 temp = i.combat()
672                                 return temp
673                             else:
674                                 print('Your ally has
     unfortunately passed away')
675                                 break
676                         else:
677                             continue
678                     else:
679                         continue
680                 print('They are not on your team')
681                 break
682             else:
683                 print('You have no allies left by
    your side')
684                 break
685
686     def team_is_alive(self):
```

```
687         for item in self.lst:
688             if item is not None:
689                 if item.is_alive():
690                     return True
691                 else:
692                     continue
693             else:
694                 continue
695         return False
696
```

```
# All the imports to be used in the program
import os
import re
import time
import random
from json import JSONDecodeError
from typing import Optional
from stdiomask import getpass
import hashlib
import json
import csv


import pygame
import math
from pygame.locals import *

from classes import user, Room, Item,
HitlerSubordinates, Allies, Team, Hitler
from caesarcipher import CaesarCipher
from random import randint

# Instantiation
# Creating Objects to assign to characters or rooms
Room_Key = Item('broadcast room key')
knife = Item('knife')
Gun = Item('gun')
poison_vial = Item('poison vial')
revolver = Item('revolver')
cyanide = Item('cyanide')
pistol = Item('pistol')
poison_dart = Item('poison dart')
dagger = Item('dagger')
sheriff_deagle = Item('deagle')
dual_pistols = Item('dual pistols')
arsenic_injections = Item('arsenic injections')

# Room Declarations
main_hall = Room("Main Hall", "This the enthralling
main hallway of the the Fuhrer himself, proceed with
caution", None)
```

```
generals_room = Room("Generals' Room", 'This is where all the strategic plans are laid out', Room_Key)
broadcast_room = Room("The Broadcast Room", "The department that makes sure Germans get their daily dose of propaganda",
                      None)
dining_hall = Room("Dining Hall", "A communal place for all the soldiers to meet, greet and eat", cyanide)
shooting_range = Room("The Shooting Range", "The range where soldiers are required to practice their aim", None)
main_hall.link_room(generals_room, "east")
generals_room.link_room(main_hall, "west")
main_hall.link_room(dining_hall, 'west')
dining_hall.link_room(main_hall, 'east')
dining_hall.link_room(broadcast_room, 'north')
broadcast_room.link_room(dining_hall, 'south')
generals_room.link_room(shooting_range, "south")
shooting_range.link_room(generals_room, "north")

# Character Declarations
General_Aladeen = HitlerSubordinates('Aladeen', 'General', 'In charge of ammunition and organized sieges', 2,
                                     'knife', 2, 4, 'kids')
Dining_Hall_Soldier = HitlerSubordinates('Sawcon', 'Private', 'In charge of the dining hall mess', 1, 'gun', 3, 3,
                                         'body')
hitler = Hitler('Adolf Hitler', 'Fuhrer', 'The Fuhrer of the Reich', None, 'cyanide', None, 15)

# Assigning objects to Characters
Dining_Hall_Soldier.change_inventory(Room_Key, True)
General_Aladeen.change_inventory(Gun, True)
hitler.add_items(poison_vial, revolver, pistol, poison_dart)

# Variable Declarations for Global Variables
```

```
team_lst = []
user1: Optional[user] = None
Team1: Optional[Team] = None
General_Aladeen_Combat: Optional[Allies] = None
Guard_Gorbachev: Optional[Allies] = None
Private_Sawcon: Optional[Allies] = None
Camp_Officer: Optional[Allies] = None

# Data Set Declaration
data = {
    'Username': None,
    'Password': None,
    'Name': None,
    'Attributes': None,
    'Dining Hall Speech': None,
    'Inventory': None,
    'Score from Brick Game': None,
    'Shooting range score': None,

}


def updateUserObject(user_data: dict):  # replaces user data to update the file (searches for same username)
    if 'Username' not in user_data:
        return

    with open('My record.json', 'r') as file:
        lines = file.readlines()

        for i in range(len(lines)):
            try:
                obj = json.loads(lines[i])
                if obj['Username'] == user_data['Username']:
                    lines[i] = json.dumps(user_data) + '\n'
                    break
            except JSONDecodeError:
                continue
```

```
103
104     with open('My record.json', 'w') as file:
105         file.writelines(lines)
106
107
108 def Add_Value_to_Data(name_, field, value):  # replaces specific field in user's data
109     with open('My record.json', 'r') as file:
110         lines = file.readlines()
111
112         for i in range(len(lines)):
113             try:
114                 obj = json.loads(lines[i])
115                 if obj['Username'] == name_:
116                     obj[field] = value
117                     lines[i] = json.dumps(obj) + '\n'
118                     break
119             except JSONDecodeError:
120                 continue
121
122     with open('My record.json', 'w') as file:
123         file.writelines(lines)
124
125
126 # General subroutines
127 def intro():
128     print('You find yourself walking down a very familiar path, wishing you could alter history in anyway possible, '
129           'after a long day at school of memorizing dates for important events in Nazi Regime you just wish you could '
130           'really live it! Suddenly you fall into a hole and find yourself in all to familiar setting in 1945')
131
132
133 def login_system():
134     # Helps clear the screen
135     def clear_screen():
```

```
136         # for mac and linux(here, os.name is 'posix
    ')
137         if os.name == 'posix':
138             _ = os.system('clear')
139         else:
140             # for windows platform
141             _ = os.system('cls')
142 
143     def Main_Menu():
144         clear_screen()
145         print('This is the main menu')
146         print("---------")
147         print()
148         print('Choose: ')
149         print('1 - Register')
150         print('2 - Login')
151         print('3 - Exit')
152         choice = None
153         while True:
154             choice = input('> ').strip()
155             if choice == '1':
156                 temp = Register()
157                 return temp
158             if choice == '2':
159                 Login()
160                 break
161             if choice == '3':
162                 exit()
163 
164     def Register():
165         clear_screen()
166         print('Register')
167         print()
168         print("---------")
169         username = None
170         while True:
171             username = input('Enter username: ').
    strip()
172             if username != '':
173                 break
174         if check_for_user(username):
```

```
175             displayUserAlreadyExistMessage()
176         else:
177             while True:
178                 password = getpass('Enter password
    : ')
179                 if password != '':
180                     break
181             while True:
182                 check_password = getpass('Re-enter
    your password: ')
183                 if check_password == password:
184                     break
185                 else:
186                     print('Password does not match')
187                     print()
188             # After username and password validation
     call subroutine to add to data file
189             Add_User_Info(username, hash_password(
    password))
190             return username
191 
192     def Login():
193         clear_screen()
194         print("LOGIN")
195         print("-----")
196         print()
197         username = input('Enter your Username: ').
    strip()
198         # Subroutine returns boolean value thus
    displays details or asks for password again
199         if check_for_user(username):
200             while True:
201                 password = getpass('Enter your
    password: ').strip()
202                 if CheckPassword(username,
    hash_password(password)):
203                     print('You are in the system!')
204                     with open('My record.json', 'r'
    ) as f:
205                         for line in f.readlines():
206                             try:
```

```
207                                     if line == '':
208                                         continue
209                                     temp = json.loads(
    line)
210                                     if temp['Username'
    ] == username:
211                                         print('These are
     your details')
212                                         print('Username
    :', temp['Username'])
213                                         print('Name:',
    temp['Name'])
214                                         print('
    Attributes:', temp['Attributes'])
215                                         print('Dining
    Hall Speech:', temp['Dining Hall Speech'])
216                                         print('Inventory
    :', temp['Inventory'])
217                                         print('Score
    from Brick Game:', temp['Score from Brick Game'])
218                                         print('Shooting
    range score:', temp['Shooting range score'])
219                                         input('Press
    enter to quit\n>')
220                                     else:
221                                         continue
222                                 except JSONDecodeError:
223                                     continue
224                     quit()
225                 else:
226                     print('Wrong password')
227                     continue
228         else:
229             print('You are not Registered!')
230             time.sleep(1)
231             Main_Menu()
232
233     def Add_User_Info(username, password):
234         d = {'Username': username}
235         d1 = {'Password': password}
236         data.update(d)
```

```
237         data.update(d1)
238         j = json.dumps(data)
239         with open('My record.json', 'a') as f:
240             f.write(j)
241             f.write('\n')
242             f.close()
243 
244     def Add_New_Field(field, element):
245         new = {field: element}
246         data.update(new)
247 
248     # Returns boolean Values for checking
249     def check_for_user(username):
250         with open('My record.json', 'r') as f:
251             for line in f.readlines():
252                 try:
253                     if line == '':
254                         continue
255                     temp = json.loads(line)
256                     if temp['Username'] == username:
257                         return True
258                     else:
259                         continue
260                 except JSONDecodeError:
261                     continue
262 
263     # Returns boolean Values for checking
264     def CheckPassword(username, password):
265         with open('My record.json', 'r') as f:
266             for line in f.readlines():
267                 try:
268                     if line == '':
269                         continue
270                     temp = json.loads(line)
271                     if not temp['Username'] ==
    username:
272                         continue
273                     elif temp['Password'] ==
    password:
274                         return True
275                     else:
```

```
                            return False
                    except JSONDecodeError:
                        continue

    def displayUserAlreadyExistMessage():
        while True:
            print()
            error = input("You Are Already
    Registered.\n\nPress (T) To Try Again:\nPress (L) To
     Login: ").lower()
            if error == 't':
                Register()
                break
            elif error == 'l':
                Login()
                break

    # Hashing module used to hash password
    def hash_password(password):
        return hashlib.sha256(str.encode(password)).
    hexdigest()

    m = Main_Menu()
    return m


def screen_clear():
    # for mac and linux(here, os.name is 'posix')
    if os.name == 'posix':
        _ = os.system('clear')
    else:
        # for windows platform
        _ = os.system('cls')


def assign_points():
    total = 8
    x1, x2, x3, x4, x5 = 0, 0, 0, 0, 0

    while total != 0:

```

```
314         try:
315             # abs makes sure the value is positive
    to avoid negative entries
316             x1 = abs(int(input("charisma: ")))
317             x2 = abs(int(input("passion: ")))
318             x3 = abs(int(input("skill: ")))
319             x4 = abs(int(input("charm: ")))
320             x5 = abs(int(input("fame: ")))
321 
322             total -= x1 + x2 + x3 + x4 + x5
323 
324         except Exception:
325 
326             print("invalid input try again")
327             # returns the function again
328             return assign_points()
329 
330         if total < 0:
331             x1 = 0
332             x2 = 0
333             x3 = 0
334             x4 = 0
335             x5 = 0
336             print("u have exceeded the limit, try
    again ")
337             return assign_points()
338 
339     # Adds the attributes and their values to json
    files to save
340     Add_Value_to_Data(user_username, 'Attributes',
341                       {'Charisma': x1, 'Passion': x2
    , 'Skill': x3, 'Charm': x4, 'Fame': x5})
342     return x1, x2, x3, x4, x5
343 
344 
345 def user_setup():
346     screen_clear()
347     print("customize your character, you have 8
    points to assign to your characters attributes,
    these points would be "
348           "distributed between passion, charisma,
```

```
348 skill, fame and charm. Any negative inputs would be
    taken as absolute"
349           " positive values")
350     charisma, passion, skill, charm, fame =
    assign_points()
351     # Uses regular expressions to validate name to
    make sure to avoid numbers or blank spaces
352     pattern = "[A-Za-z]+"
353     name = input("name: ")
354     while not re.fullmatch(pattern, name):
355         name = input("invalid name try again: ")
356     global user1
357     # Creates user object using attributes from user
     input
358     user1 = user(name, charisma, passion, skill,
    charm, fame)
359     print("your charisma is:", user1.charisma)
360     print("your passion is:", user1.passion)
361     print("your skill is:", user1.skill)
362     print("your charm is:", user1.charm)
363     print('your fame is:', user1.fame)
364     print("your name is :", user1.name)
365     Add_Value_to_Data(user_username, 'Name', name)
366
367
368 def current_user_stats():
369     print("your charisma is:", user1.charisma)
370     print("your passion is:", user1.passion)
371     print("your skill is:", user1.skill)
372     print("your charm is:", user1.charm)
373     print('your current hp is:', user1.hp)
374     # So that the object itself isn't printed
375     lst = []
376     for i in user1.inventory:
377         temp = i.name
378         lst.append(temp)
379     print('your current inventory is:', lst)
380
381
382 def menu(character, room):
383     # Passing character and room objects into menu
```

```
383 subroutine to call their methods and alter their
    attributes
384     # accordingly
385     selection = 0
386     tries = 0
387     while selection != '6' and tries < 5:
388         print("you have between 6 options:\n1) Steal
     from", character.name, "\n2) Search",
389               room.name, "\n3) Salute", character.
    name, "\n4) Fight", character.name, '\n5) Compliment
    ',
390               character.name, "\n6) Exit Menu")
391         selection = input("> ")
392         if selection == "1":
393             character.steal(user1)
394             current_user_stats()
395             tries += 1
396             selection = 0
397         elif selection == "2":
398             room.search_room(user1)
399             current_user_stats()
400             tries += 1
401             selection = 0
402         elif selection == "3":
403             character.salute(user1)
404             current_user_stats()
405             tries += 1
406             selection = 0
407         elif selection == "4":
408             character.fight(user1)
409             current_user_stats()
410             tries += 1
411             selection = 0
412         elif selection == '5':
413             character.compliment(user1)
414             current_user_stats()
415             tries += 1
416             selection = 0
417         elif selection == '6':
418             current_user_stats()
419             break
```

```
420         else:
421             continue
422 
423 
424 def binary_decision_function(expression, fn1, fn2):
425     selection = 0
426     print(expression)
427     while selection not in ('1', '2'):
428         try:
429             selection = input("> ")
430             if selection == '1':
431                 fn1()
432             elif selection == '2':
433                 fn2()
434         except Exception:
435             pass
436 
437 
438 def binary_decision_general(user_defined_input,
    input1, input2):
439     selection = 0
440     while selection not in ('1', '2'):
441         try:
442             selection = int(input(user_defined_input
    ))
443             if selection == '1':
444                 print(input1)
445             elif selection == '2':
446                 print(input2)
447                 pass
448         except Exception:
449             pass
450 
451 
452 def validation():
453     temp = False
454     input1 = 0
455     while not temp:
456         try:
457             input1 = int(input("Enter (integer): "))
458             temp = True
```

```
459         except Exception:
460             pass
461         print("try again")
462     return input1
463 
464 
465 # Using the caesar cipher module to encode messages
466 def level1(time_):
467     cipher1 = CaesarCipher('Fuhrer', offset=14)
468     cipher2 = CaesarCipher('War', offset=14)
469     cipher3 = CaesarCipher('Rations', offset=14)
470     cipher4 = CaesarCipher('Revolution', offset=14)
471     cipher5 = CaesarCipher('Economy', offset=14)
472 
473     print('your offset is 14')
474     print(cipher1.encoded, cipher2.encoded, cipher3.
    encoded, cipher4.encoded, cipher5.encoded)
475     timer(time_)
476     screen_clear()
477 
478 
479 def level2(time_):
480     lst = []
481     for x in range(5):
482         lst.append(random.randint(1, 25))
483 
484     cipher1 = CaesarCipher('Fuhrer', offset=lst[0])
485     cipher2 = CaesarCipher('War', offset=lst[1])
486     cipher3 = CaesarCipher('Rations', offset=lst[2])
487     cipher4 = CaesarCipher('Revolution', offset=lst[
    3])
488     cipher5 = CaesarCipher('Economy', offset=lst[4])
489 
490     for x in range(len(lst)):
491         print('your offset is:', lst[x])
492 
493     print(cipher1.encoded, cipher2.encoded, cipher3.
    encoded, cipher4.encoded, cipher5.encoded)
494     timer(time_)
495     screen_clear()
496 
```

```
497
498 def level3(time_):
499     lst = []
500     for x in range(5):
501         lst.append(random.randint(1, 25))
502
503     cipher1 = CaesarCipher('Fuhrer', offset=lst[0])
504     cipher2 = CaesarCipher('War', offset=lst[1])
505     cipher3 = CaesarCipher('Rations', offset=lst[2])
506     cipher4 = CaesarCipher('Revolution', offset=lst[
    3])
507     cipher5 = CaesarCipher('Economy', offset=lst[4])
508     print("a = 1 and ' = 0")
509
510     for x in range(len(lst)):
511         z = lst[x] // 10
512         y = lst[x] % 10
513         print('your offset is:', chr(z + 96), chr(y
     + 96))
514
515     print(cipher1.encoded, cipher2.encoded, cipher3.
    encoded, cipher4.encoded, cipher5.encoded)
516     timer(time_)
517     screen_clear()
518
519
520 # subroutine that keeps track of active time on
    terminal
521 def timer(amount_of_time):
522     t = amount_of_time
523     temp = True
524     while temp:
525         while t != 0:
526             mins = t // 60
527             secs = t % 60
528             timer_ = '{:02d}:{:02d}'.format(mins,
    secs)
529             print(f"\r{timer_}", end="", flush=True)
530             time.sleep(1)
531             t -= 1
532         temp = False
```

```
533
534
535 # Subroutine to transition between rooms by passing
    the room objects and room subroutines
536 def room_transition(rt_input1, rt_input2, rt_input3
    , rt_function1, rt_function2):
537     current_state = True
538     while current_state:
539         print("\n")
540         rt_input3.get_details()
541         print("Where do you want to go: ")
542         command = input("> ")
543         current_room = rt_input3.move(command.lower
    ().strip())
544         if current_room == rt_input1:
545             rt_function1()
546             current_state = False
547         elif current_room == rt_input2:
548             rt_function2()
549             current_state = False
550
551
552 # Mini-Game subroutines
553 def aimTrainer(time_limit):
554     pygame.init()
555     # Constants to be set
556     WIDTH = 900
557     HEIGHT = 600
558     SCREEN = pygame.display.set_mode((WIDTH, HEIGHT
    ))
559     text_x = 10
560     text_y = 10
561     time_limit = time_limit
562
563     # Colors
564     black = (0, 0, 0)
565     white = (255, 255, 255)
566     purple = (128, 0, 128)
567     grey = (128, 128, 128)
568     sky = (0, 0, 220)
569     blue = (85, 206, 255)
```

```
570     orange = (255, 127, 80)
571     red = (200, 0, 0)
572     light_red = (255, 0, 0)
573     green = (0, 200, 0)
574     light_green = (0, 255, 0)
575     colors = [white, grey, purple, sky, blue, orange
    , red, light_red, green, light_green]
576 
577     clock = pygame.time.Clock()  # To set the frame
    rate
578 
579     # Changing variables
580 
581     score = 0
582     start_time = time.time()
583 
584     # Setting up screen and circles
585     font = pygame.font.SysFont('verdana', 32)
586     cx = random.randint(100, WIDTH - 100)
587     cy = random.randint(100, HEIGHT - 100)
588     width_of_circle = random.randint(14, 20)
589     pygame.draw.circle(SCREEN, random.choice(colors
    ), (cx, cy), width_of_circle)
590 
591     # Function to show score
592     def show_score(z, y):
593         score_value = font.render('Score: ' + str(
    score), True, (255, 255, 255))
594         SCREEN.blit(score_value, (z, y))
595 
596     # Main loop
597     while True:
598         for event in pygame.event.get():
599             if event.type == pygame.QUIT:
600                 pygame.quit()
601                 return score
602         # Keep a check of time elapsed since start
    of game
603         elapsed_time = time.time() - start_time
604         # check if time is over, then finish the
    subroutine and return the score
```

```
605         if elapsed_time > time_limit:
606             print('game over')
607             pygame.quit()
608             return score
609 
610         x = pygame.mouse.get_pos()[0]
611         y = pygame.mouse.get_pos()[1]
612         click = pygame.mouse.get_pressed()
613         # Using maths to keep track of circle and
    mouse collision
614 
615         sqx = (x - cx) ** 2
616         sqy = (y - cy) ** 2
617 
618         if math.sqrt(sqx + sqy) < width_of_circle
    and click[0] == 1:
619             SCREEN.fill(black)  # Reset the screen
620             cx = random.randint(20, WIDTH - 20)
621             cy = random.randint(20, HEIGHT - 20)
622             width_of_circle = random.randint(14, 20)
623             pygame.draw.circle(SCREEN, random.choice
    (colors), (cx, cy), width_of_circle)
624             score += 1
625 
626         show_score(text_x, text_y)
627         pygame.display.update()
628         clock.tick()
629 
630 
631 def breakout(user_lives, speed):
632     pygame.init()
633     # Setting constants
634     SCREEN_WIDTH = 600
635     SCREEN_HEIGHT = 600
636 
637     SCREEN = pygame.display.set_mode((SCREEN_WIDTH,
    SCREEN_HEIGHT))
638 
639     pygame.display.set_caption('BreakMe')
640 
641     font = pygame.font.SysFont('Constantia', 30)
```

```
    # background color
    bg = (234, 218, 184)

    # brick colors
    brick_red = (242, 85, 96)
    brick_green = (86, 174, 87)
    brick_blue = (89, 177, 232)
    # paddle colors
    paddle_color = (142, 135, 123)
    paddle_outline = (100, 100, 100)
    text_colour = (78, 81, 139)

    # game variables
    COLS = 6
    ROWS = 6
    clock = pygame.time.Clock()
    FPS = 60
    live_ball = False
    game_over = 0
    lives = user_lives
    ball_speed = speed

    def write_text(text, _font, text_col, x, y):
        image = _font.render(text, True, text_col)
        SCREEN.blit(image, (x, y))

    # classes
    class Wall:
        def __init__(self):
            self.brick = None
            self.width = SCREEN_WIDTH // COLS
            self.height = 50

        def create_wall(self):
            self.brick = []
            strength = None
            # empty list for single block
            brick_individual = []
            for row in range(ROWS):
                # reset the block row list
                brick_row = []
```

```
683                 # iterate through each col in the
    row
684                 for col in range(COLS):
685                     # produce x and y positions and
    create rectangle from these positions
686                     brick_x = col * self.width
687                     brick_y = row * self.height
688                     rect = pygame.Rect(brick_x,
    brick_y, self.width, self.height)
689                     # assign brick strength based on
     the rows:
690                     if row < 2:
691                         strength = 3
692                     elif row < 4:
693                         strength = 2
694                     elif row < 6:
695                         strength = 1
696                     # creating list to store
    rectangle and its color
697                     brick_individual = [rect,
    strength]
698                     # appending individual brick to
    the brick row
699                     brick_row.append(
    brick_individual)
700                 # adding row to the whole list of
    the blocks
701                 self.brick.append(brick_row)
702
703         def draw_wall(self):
704             brick_color = None
705             for row in self.brick:
706                 for brick in row:
707                     # set a color based on brick
    level strength
708                     if brick[1] == 3:
709                         brick_color = brick_blue
710                     elif brick[1] == 2:
711                         brick_color = brick_green
712                     elif brick[1] == 1:
713                         brick_color = brick_red
```

```
714                     # drawing the bricks
715                     pygame.draw.rect(SCREEN,
    brick_color, brick[0])
716                     # to draw a border for each
    brick to be differentiated
717                     pygame.draw.rect(SCREEN, bg, (
    brick[0]), 1)
718 
719     class Paddle:
720         def __init__(self):
721             # setting paddle variables
722             self.height = 20
723             self.width = int(SCREEN_WIDTH / COLS)
724             self.x = int((SCREEN_WIDTH / 2) - (self.
    width / 2))
725             self.y = SCREEN_HEIGHT - (self.height *
    2)
726             self.speed = 10
727             self.rect = Rect(self.x, self.y, self.
    width, self.height)
728             self.direction = 0
729 
730         def move(self):
731             # resets the direction of the movement
732             self.direction = 0
733             key = pygame.key.get_pressed()
734             if key[pygame.K_LEFT] and self.rect.left
     > 0:
735                 self.rect.x -= self.speed
736                 self.direction = -1
737             if key[pygame.K_RIGHT] and self.rect.
    right < SCREEN_WIDTH:
738                 self.rect.x += self.speed
739                 self.direction = 1
740 
741         def draw(self):
742             pygame.draw.rect(SCREEN, paddle_color,
    self.rect)
743             pygame.draw.rect(SCREEN, paddle_outline
    , self.rect, 3)
744 
```

```
745         def reset(self):
746             # to reset the paddle to original
    dimensions and positions for game reset
747             self.height = 20
748             self.width = int(SCREEN_WIDTH / COLS)
749             self.x = int((SCREEN_WIDTH / 2) - (self.
    width / 2))
750             self.y = SCREEN_HEIGHT - (self.height *
    2)
751             self.speed = 10
752             self.rect = Rect(self.x, self.y, self.
    width, self.height)
753             self.direction = 0
754 
755     class Ball:
756         def __init__(self, x, y, _speed):
757             self.radius = 10
758             self.x = x - self.radius
759             self.y = y
760             self.rect = Rect(self.x, self.y, self.
    radius * 2, self.radius * 2)
761             self.speed_x = _speed
762             self.speed_y = -_speed
763             self.max_speed = _speed + 1
764             self.game_over = 0
765 
766         def move(self, _lives):
767             collision_threshold = 5
768             player_lives = _lives
769 
770             # checking for collision with walls -
    assuming wall has been destroyed completely
771             wall_destroyed = 1
772             row_counter = 0
773             for row in wall.brick:
774                 item_counter = 0
775                 for item in row:
776                     # checking for collision
777                     if self.rect.colliderect(item[0
    ]):
778                         # collision from above
```

```
779                         if abs(self.rect.bottom -
    item[0].top) < collision_threshold and self.speed_y
     > 0:
780                             self.speed_y *= -1
781                         # collision from below
782                         if abs(self.rect.top - item[
    0].bottom) < collision_threshold and self.speed_y <
    0:
783                             self.speed_y *= -1
784                         # collision from the right
785                         if abs(self.rect.right -
    item[0].left) < collision_threshold and self.speed_x
     > 0:
786                             self.speed_x *= -1
787                         # collision from the left
788                         if abs(self.rect.left - item
    [0].right) < collision_threshold and self.speed_x <
    0:
789                             self.speed_x *= -1
790                         # reducing brick strength
791                         if wall.brick[row_counter][
    item_counter][1] > 1:
792                             wall.brick[row_counter][
    item_counter][1] -= 1
793                         else:
794                             wall.brick[row_counter][
    item_counter][0] = (0, 0, 0, 0)
795                         # check if any brick exists
796                         if wall.brick[row_counter][
    item_counter][0] != (0, 0, 0, 0):
797                             wall_destroyed = 0
798                         # increment item counter
799                         item_counter += 1
800                     # increment row counter
801                     row_counter += 1
802                 # checking if wall is destroyed after
    going through all bricks
803                 if wall_destroyed == 1:
804                     self.game_over = 1
805 
806                 if self.rect.left < 0 or self.rect.right
```

```
806  > SCREEN_WIDTH:
807                  self.speed_x *= -1
808              if self.rect.top < 0:
809                  self.speed_y *= -1
810              if self.rect.bottom > SCREEN_HEIGHT:
811                  self.game_over = -1
812                  player_lives -= 1
813              # collision with paddle
814              if self.rect.colliderect(user_paddle):
815                  # check for only collisions from top
    :
816                  if (abs(self.rect.bottom -
    user_paddle.rect.top) < collision_threshold) and (
    self.speed_y > 0):
817                      self.speed_y *= -1
818                      self.speed_x += user_paddle.
    direction
819                      if self.speed_x > self.max_speed
    :
820                          self.speed_x = self.
    max_speed
821                      elif self.speed_x < 0 and self.
    speed_x < - self.max_speed:
822                          self.speed_x = - self.
    max_speed
823                  else:
824                      self.speed_x *= -1
825
826              self.rect.x += self.speed_x
827              self.rect.y += self.speed_y
828
829              return self.game_over, player_lives
830
831          def draw(self):
832              pygame.draw.circle(SCREEN, paddle_color
    , ((self.rect.x + self.radius), (self.rect.y + self.
    radius)),
833                                 self.radius)
834              pygame.draw.circle(SCREEN,
    paddle_outline, ((self.rect.x + self.radius), (self.
    rect.y + self.radius)),
```

```
835                                     self.radius, 3)
836
837         def reset(self, x, y, _speed):
838             self.radius = 10
839             self.x = x - self.radius
840             self.y = y
841             self.rect = Rect(self.x, self.y, self.
    radius * 2, self.radius * 2)
842             self.speed_x = _speed
843             self.speed_y = -_speed
844             self.max_speed = _speed + 1
845             self.game_over = 0
846
847     # creating objects
848     wall = Wall()
849     wall.create_wall()
850     user_paddle = Paddle()
851     ball = Ball(user_paddle.x + (user_paddle.width
     // 2), user_paddle.y - user_paddle.height,
    ball_speed)
852
853     run = True
854
855     while run and lives > 0:
856         clock.tick(FPS)
857         SCREEN.fill(bg)
858
859         # drawing the wall
860         wall.draw_wall()
861         user_paddle.draw()
862         ball.draw()
863         if live_ball:
864             user_paddle.move()
865             game_over, lives = ball.move(lives)
866             if game_over != 0:
867                 live_ball = False
868
869         # player instructions
870         if not live_ball:
871             if game_over == 0:
872                 write_text('Click anywhere to start'
```

```
872 , font, text_colour, 150, SCREEN_HEIGHT // 2 + 100)
873             elif game_over == 1:
874                 write_text('You won', font,
    text_colour, 250, SCREEN_HEIGHT // 2 + 50)
875                 pygame.quit()
876                 return lives
877             elif game_over == -1:
878                 write_text('You lost', font,
    text_colour, 250, SCREEN_HEIGHT // 2)
879                 write_text(('You have ' + str(lives
    ) + ' try(s) left'), font, text_colour, 165,
    SCREEN_HEIGHT // 2 + 50)
880                 write_text('Click anywhere to start'
    , font, text_colour, 150, SCREEN_HEIGHT // 2 + 100)
881 
882         for event in pygame.event.get():
883             if event.type == pygame.QUIT:
884                 #lives = 0
885                 run = False
886                 pygame.quit()
887                 return lives
888             if event.type == pygame.MOUSEBUTTONDOWN
    and live_ball is False:
889                 live_ball = True
890                 ball.reset(user_paddle.x + (
    user_paddle.width // 2), user_paddle.y - user_paddle
    .height, ball_speed)
891                 user_paddle.reset()
892                 wall.create_wall()
893 
894         pygame.display.update()
895 
896     pygame.quit()
897     return lives
898 
899 
900 # Story Subroutines
901 
902 def rallying_speech():
903     time_amount = 180
904     print('as you make up your mind to give the
```

```
904 speech a small envelope is shoved into your pocket')
905     print('it contains the following words: ')
906     print('decrypt this message to find out what
    stirs soldiers towards rebellion and include it in
    your speech')
907     if user1.skill <= 0:
908         time_amount = 160
909     elif user1.skill > 2:
910         time_amount = 200
911 
912     if user1.passion <= 0:
913         level3(time_amount)
914 
915     elif user1.passion > 2:
916         level1(time_amount)
917 
918     else:
919         level2(time_amount)
920 
921     temp = 0
922     user_speech = input('enter your speech:\n> ')
923     user_speech.split(' ')
924     # Uses list operations to validate user input to
     given words
925     lst = ['fuhrer', 'war', 'revolution', 'rations'
    , 'economy']
926     for x in range(len(lst)):
927         if lst[x] in user_speech:
928             temp += 1
929     if temp == 5:
930         user1.change_charisma(True)
931         user1.change_fame(True)
932         user1.change_charm(True)
933         print('your speech caused the whole infantry
     to erupt in applause, you are carried on their
    soldiers as a'
934               ' messiah')
935     elif 3 <= temp < 5:
936         user1.change_fame(True)
937         user1.change_charisma(True)
938         print('your speech caused a disturbance
```

```
938 among most soldiers as they nod their head in
    agreement')
939
940     elif 1 <= temp < 3:
941         user1.change_fame(True)
942         print('your speech arouses a few suspicious
    looks around but you also see a few approving looks'
    )
943     else:
944         user1.change_charisma(False)
945         user1.change_fame(False)
946         print('you are booed off the stage for your
    speech and you look for an escape before anyone
    reports your name')
947     # Adds to speech to the json file
948     Add_Value_to_Data(user_username, 'Dining Hall
    Speech', user_speech)
949
950
951 def class_creation(name):
952     # Creates characters for final boss battle
953     global General_Aladeen_Combat, Private_Sawcon,
    Guard_Gorbachev
954     if name == 'general aladeen':
955         General_Aladeen_Combat = Allies('general
    aladeen', 'The General in charge of forming
    strategies',
956                                         'revolver',
    'was never given any credit')
957         General_Aladeen_Combat.add_items(dagger)
958     elif name == 'private sawcon':
959         Private_Sawcon = Allies('private sawcon', '
    The soldier in charge of rations and motivation',
960                                 'pistol', "feels
    like the reich would be better off in someone else's
     hands")
961         Private_Sawcon.add_items(sheriff_deagle)
962     elif name == 'guard gorbachev':
963         Guard_Gorbachev = Allies('guard gorbachev',
    'Guard in charge of the shooting range and
    ammunition',
```

```
964                                     'poison vial', '
    has been part of the reich for 20 years now and
    never gotten above the'
965                                                     '
    rank of a guard')
966         Guard_Gorbachev.add_items(dual_pistols)
967
968     return General_Aladeen_Combat, Private_Sawcon,
    Guard_Gorbachev
969
970
971 def Broadcast_Room():
972     while Room_Key in user1.inventory:
973         while not broadcast_room.room_visited:
974             print('You find the Broadcast Room
    deserted and empty, and you find this as your
    opportunity '
975                   'to put the final nail in the
    coffin')
976             print('For every life you conserve you
    get to make an ally from within the regime, anyone
    memorable you '
977                   'have met throughout the game
    such as "general x"...provided you remember their
    name and you have '
978                   'left an impressionable mark upon
     them')
979             time.sleep(2)
980             number_of_lives = 4
981             if user1.skill < 0:
982                 number_of_lives = 3
983             elif user1.skill > 2:
984                 number_of_lives = 5
985             speed = 3
986             if user1.charisma < 0:
987                 speed = 4
988             elif user1.charisma > 2:
989                 speed = 2
990             score = breakout(number_of_lives, speed
    )
991             print('your score:', score)
```

```
            global team_lst, Camp_Officer, Team1
            if score == 5:
                print('You get to urge 3 comrades of yours to join your resistance against the fuhrer, however get 4 '
                      'tries broadcast their name on media and they will be on your side however get their name wrong '
                      'and it would be very hard to convince them after')
                broadcast1 = input("Broadcast Name 1: \n>").lower().strip()
                broadcast2 = input("Broadcast Name 2: \n>").lower().strip()
                broadcast3 = input("Broadcast Name 3: \n>").lower().strip()
                broadcast4 = input("Broadcast Name 4: \n>").lower().strip()
                ally_list = ['general aladeen', 'private sawcon', 'guard gorbachev']
                if broadcast1 in ally_list:
                    ally_list.remove(broadcast1)
                    a, b, c = class_creation(broadcast1)
                else:
                    continue
                if broadcast2 in ally_list:
                    ally_list.remove(broadcast2)
                    a, b, c = class_creation(broadcast2)
                else:
                    continue
                if broadcast3 in ally_list:
                    ally_list.remove(broadcast3)
                    a, b, c = class_creation(broadcast3)
                else:
                    continue
                run = True
                if len(ally_list) == 0:
                    run = False
```

```
                while run:
                    if broadcast4 in ally_list:
                        ally_list.remove(broadcast4
    )
                        class_creation(broadcast4)
                        a, b, c = class_creation(
    broadcast4)
                        run = False
                    else:
                        run = False
                print("For being exceptional at
    this task you also awarded with an opportunity to
    name a person you "
                      "hope would join your
    rebellion, a person in control of the camps for you
     don't agree with the "
                      'practice and hope to free
    them')
                special = input("> ").strip().lower
    ()
                if special == 'rudolf höss' or
    special == 'theodor eicke' or special == 'heinrich
    himmler' \
                        or special == 'rudolf hoess
    ':
                    print('You have chosen the
    right man for this job')
                    Camp_Officer = Allies(special,
    'camp officer in charge of the camps', 'poison dart
    ',
                                          "does not
     agree with the practices but has to his job")
                    Camp_Officer.add_items(
    arsenic_injections)
                else:
                    print('Unfortunately that is a
    person who would not be able to help')
                Team1 = Team(a, b, c, Camp_Officer)
            elif score == 4:
                print('You get to urge 3 comrades
    of yours to join your resistance against the fuhrer
```

```
1042 , however get 4 '
1043                         'tries broadcast their name
     on media and they will be on your side however get
     their name wrong '
1044                         'and it would be very hard to
      convince them after')
1045                 broadcast1 = input("Broadcast Name
     1: \n>").lower().strip()
1046                 broadcast2 = input("Broadcast Name
     2: \n>").lower().strip()
1047                 broadcast3 = input("Broadcast Name
     3: \n>").lower().strip()
1048                 broadcast4 = input("Broadcast Name
     4: \n>").lower().strip()
1049                 ally_list = ['general aladeen', '
     private sawcon', 'guard gorbachev']
1050                 if broadcast1 in ally_list:
1051                     ally_list.remove(broadcast1)
1052                     a, b, c = class_creation(
     broadcast1)
1053                 else:
1054                     continue
1055                 if broadcast2 in ally_list:
1056                     ally_list.remove(broadcast2)
1057                     a, b, c = class_creation(
     broadcast2)
1058                 else:
1059                     continue
1060                 if broadcast3 in ally_list:
1061                     ally_list.remove(broadcast3)
1062                     a, b, c = class_creation(
     broadcast3)
1063                 else:
1064                     continue
1065                 run = True
1066                 if len(ally_list) == 0:
1067                     run = False
1068                 while run:
1069                     if broadcast4 in ally_list:
1070                         ally_list.remove(broadcast4
     )
```

```
                            a, b, c = class_creation(
    broadcast4)
                            run = False
                        else:
                            run = False
                    Team1 = Team(a, b, c)
                elif score == 3:
                    print('You get to urge 3 comrades
    of yours to join your resistance against the fuhrer
    , get 3 tries to'
                          ' broadcast their name on
    media and they will be on your side however get
    their name wrong and '
                          'it would be very hard to
    convince them after')
                    broadcast1 = input("Broadcast Name
    1: \n>").lower().strip()
                    broadcast2 = input("Broadcast Name
    2: \n>").lower().strip()
                    broadcast3 = input("Broadcast Name
    3: \n>").lower().strip()
                    ally_list = ['general aladeen', '
    private sawcon', 'guard gorbachev']
                    if broadcast1 in ally_list:
                        ally_list.remove(broadcast1)
                        a, b, c = class_creation(
    broadcast1)
                    else:
                        continue
                    if broadcast2 in ally_list:
                        ally_list.remove(broadcast2)
                        a, b, c = class_creation(
    broadcast2)
                    else:
                        continue
                    if broadcast3 in ally_list:
                        ally_list.remove(broadcast3)
                        a, b, c = class_creation(
    broadcast3)
                    else:
                        continue
```

```
                Team1 = Team(a, b, c)
            elif score == 2:
                print('You get to urge 2 comrades
    of yours to join your resistance against the fuhrer
    , get 2 tries to '
                      'broadcast their name on
    media and they will be on your side however get
    their name wrong'
                      'and it would be very hard to
     convince them after')
                broadcast1 = input("Broadcast Name
    1: \n>").lower().strip()
                broadcast2 = input("Broadcast Name
    2: \n>").lower().strip()
                ally_list = ['general aladeen', '
    private sawcon', 'guard gorbachev']
                if broadcast1 in ally_list:
                    ally_list.remove(broadcast1)
                    a, b, c = class_creation(
    broadcast1)
                else:
                    continue
                if broadcast2 in ally_list:
                    ally_list.remove(broadcast2)
                    a, b, c = class_creation(
    broadcast2)
                else:
                    continue
                Team1 = Team(a, b, c)
            elif score == 1:
                print('You get to urge a comrade of
     yours to join your resistance against the fuhrer,
    get a '
                      'try to broadcast their name
    on media and they will be on your side however get
    their name wrong'
                      'and it would be very hard to
     convince them after')
                broadcast1 = input("Broadcast Name
    1: \n>").lower().strip()
                ally_list = ['general aladeen', '
```

```
1123 private sawcon', 'guard gorbachev']
1124                 if broadcast1 in ally_list:
1125                     ally_list.remove(broadcast1)
1126                     a, b, c = class_creation(
     broadcast1)
1127                 else:
1128                     continue
1129                 Team1 = Team(a, b, c)
1130             else:
1131                 print('You could not complete the
     game ')
1132                 print('Thus you will now face the
     Fuhrer alone')
1133                 Team1 = Team(General_Aladeen_Combat
     , Guard_Gorbachev, Private_Sawcon)
1134             team_lst = [General_Aladeen_Combat,
     Private_Sawcon, Camp_Officer, Guard_Gorbachev]
1135             Add_Value_to_Data(user_username, 'Score
      from Brick Game', score)
1136             broadcast_room.change_room_status()
1137             room_transition(dining_hall, None,
     broadcast_room, Dining_Hall, None)
1138             user1.change_inventory(Room_Key, False)
1139         print('You have been here before')
1140         room_transition(dining_hall, None,
     broadcast_room, Dining_Hall, None)
1141     print('You do not have the key to this room')
1142     room_transition(dining_hall, None,
     broadcast_room, Dining_Hall, None)
1143
1144
1145 def Main__Hall():
1146     global main_hall
1147     while True:
1148         while not main_hall.room_visited:
1149             print('You are walking down the path
     when a soldier in a stunning black uniform walks up
      to u and says:'
1150                   '"Ah', user1.name, 'what are you
     doing here, General Aladeen is waiting for you"')
1151             print('While the soldier accompanies
```

```
along you decide to engage in small talk. The soldier is unaware and '
                  'is about to reveal crucial information about the general however you notice a scintillating object')
            print('You have to make a choice if you would rather \n1) choose to pickpocket the mysterious object '
                  '\n2) listen to the crucial information')
            selection = 0
            while selection not in ('1', '2'):
                try:
                    selection = input("> ")
                    if selection == '1':
                        print('You decide to pickpocket the item and quickly escape the conversation to avoid'
                              ' being detected')
                        user1.change_inventory(knife, True)
                    elif selection == '2':
                        print('[Soldier]: I have heard he has an absurd fear when it comes to knives, however if you do'
                              ' want to please him do mention his kids. I have also heard he has a gun ')
                except Exception:
                    pass
            main_hall.change_room_status()
            room_transition(generals_room, dining_hall, main_hall, Generals_Room, Dining_Hall)
        print('You are back in the main hall where would u want go now')
        if generals_room.room_visited and shooting_range.room_visited and dining_hall.room_visited and \
                broadcast_room.room_visited:
            print('You have visited all rooms and are confronted by Hitler')
```

```
1174             print('He has finally caught up to you
     and confronts you about all of your shenanigans')
1175             for item in team_lst:
1176                 if item is not None:
1177                     item.introduction()
1178                 else:
1179                     continue
1180             user1.change_inventory(Room_Key, False)
1181             while hitler.is_alive():
1182                 print('Time to battle!')
1183                 decision = None
1184                 if Team1.team_is_alive():
1185                     while decision not in ['y', 'a'
     ]:
1186                         decision = input(
1187                             'Do you want to battle
     Hitler or do you want your ally to battle:\n1)y-you
     \n2)a-ally\n>') \
1188                             .strip().lower()
1189                     if decision == 'y':
1190                         hit = user1.combat()
1191                         hitler.got_hit(hit)
1192                     elif decision == 'a':
1193                         hit = Team1.team_combat()
1194                         hitler.got_hit(hit)
1195                     hitler.combat(user1, team_lst[0
     ], team_lst[1], team_lst[2], team_lst[3])
1196                     if not user1.is_alive():
1197                         print('You die game over')
1198                         print('Hitler defeats you')
1199                         input('Press enter to exit\
     n>')
1200                         exit()
1201                 else:
1202                     print('You do not have any
     allies left')
1203                     hit = user1.combat()
1204                     hitler.got_hit(hit)
1205                     hitler.combat(user1)
1206                     if not user1.is_alive():
1207                         print('You die game over')
```

```
1208                          print('Hitler defeats you')
1209                          input('Press enter to exit\
     n>')
1210                          exit()
1211
1212              hitler_maze_run()
1213              print('Game Over You Win')
1214              exit()
1215          else:
1216              room_transition(generals_room,
     dining_hall, main_hall, Generals_Room, Dining_Hall)
1217
1218
1219 def Generals_Room():
1220     while not generals_room.room_visited:
1221         print("You enter the room and are greeted
     by everyone")
1222         print("u say... greetings general...")
1223         user_test1 = input("> ")
1224         if user_test1.lower() == General_Aladeen.
     name.lower():
1225             print("[General]: I see you respect me"
     )
1226         else:
1227             user1.change_charisma(False)
1228             print("[General]: That's not my name
     soldier")
1229         print('[General]: Anyways moving on... we
     need to decide which ally to send aid to')
1230         user_test2 = input("> ")
1231         allies = ['italy', 'japan']
1232         if user_test2.lower() in allies:
1233             print("[General]: Good decision")
1234             user1.change_passion(True)
1235         else:
1236             print("[General]: They not our ally..."
     )
1237             user1.change_passion(False)
1238         print("[General]: Finally the topic to
     discuss we need a plan to attack a country...")
1239         user_decision1 = input("> ")
```

```
1240         if user_decision1.lower() == "russia":
1241             print("[General]: That might be
     interesting provided russia winters have just
     passed")
1242             user1.change_charisma(True)
1243         elif user_decision1.lower() == "belgium":
1244             print("[General]: Ah as per the
     original plan one would say")
1245             user1.change_passion(True)
1246         else:
1247             print('[General]: Why would you attack
     them??')
1248             user1.change_passion(False)
1249         print("[General]: Is there any particular
     way you would like to lay siege?")
1250         user_test3 = input("> ")
1251         if user_test3.lower() == 'blitzkrieg':
1252             user1.change_passion(True)
1253             print('[General]: Original to the plan'
     )
1254         else:
1255             print('[General]: Hmm i really doubt
     that would work but thank you for your input
     regardless')
1256         print('[General]: You are free to go
     soldier however you might soon want to head to the
     shooting range to the '
1257               'right for daily practice')
1258         current_user_stats()
1259         print('You need to search the room till you
      find a key to progress')
1260         menu(General_Aladeen, generals_room)
1261         generals_room.change_room_status()
1262         room_transition(main_hall, shooting_range,
     generals_room, Main__Hall, Shooting_Range)
1263     print('You have been here before')
1264     room_transition(main_hall, shooting_range,
     generals_room, Main__Hall, Shooting_Range)
1265
1266
1267 def Dining_Hall():
```

```
1268     while not dining_hall.room_visited:
1269         if not generals_room.room_visited:
1270             print('You enter into a rather quiet
     room ')
1271             print('Yet another soldier reminds you
     that the general is waiting for you')
1272             room_transition(main_hall,
     broadcast_room, dining_hall, Main__Hall,
     Broadcast_Room)
1273         if not shooting_range.room_visited:
1274             print('The general has informed you to
     go practice at the shooting range first it is
     recommended '
1275                   'you proceed accordingly')
1276             room_transition(main_hall,
     broadcast_room, dining_hall, Main__Hall,
     Broadcast_Room)
1277         else:
1278             print('[Private Sawcon]: Oye', user1.
     name, 'we be hearing')
1279             print('[Private Sawcon]: That you had a
      meeting with good ol Aladeen')
1280             print('[Private Sawcon]: you think you
     are better than us?')
1281             print('You are given a choice')
1282             print('Either:\n1) Bad mouth the
     general or \n2) stand up for yourself')
1283             binary_decision_function("choose an
     option:", user1.change_charisma, user1.
     change_passion)
1284             print('[Private Sawcon]: "I-')
1285             print("As the  soldier is about to
     reply to you, there is an announcement about daily
     rations for "
1286                   "soldiers which causes a
     commotion")
1287             print("do you take this opportunity to
     \n1) make a rallying speech or \n2) use it as an
     distraction to "
1288                   "find an essential tool next to
     the kitchen")
```

```
1289             selection = 0
1290             print("make a choice: ")
1291             while selection not in ('1', '2'):
1292                 try:
1293                     selection = input("> ")
1294                     if selection == '1':
1295                         rallying_speech()
1296                     elif selection == '2':
1297                         user1.change_inventory(
     knife, True)
1298                         print('you scour the room
     and successfully find a mini knife')
1299                 except Exception:
1300                     pass
1301             current_user_stats()
1302             menu(Dining_Hall_Soldier, dining_hall)
1303             dining_hall.change_room_status()
1304             room_transition(main_hall,
     broadcast_room, dining_hall, Main__Hall,
     Broadcast_Room)
1305     print('You have been here already')
1306     room_transition(main_hall, broadcast_room,
     dining_hall, Main__Hall, Broadcast_Room)
1307
1308
1309 def word_check(num):
1310     word = None
1311     pattern = "[A-Za-z]+"
1312     for x in range(num):
1313         while not re.fullmatch(pattern, word):
1314             word = input("Is this your input to the
      masses? Pathetic, invalid input try again: ")
1315
1316
1317 def Shooting_Range():
1318     if shooting_range.room_visited:
1319         user1.change_skill(False)
1320     print("[Guard Gorbachev]: Welcome to the
     shooting range")
1321     print("The guard at the shooting range seems
     bored, he has decided to play a game with you")
```

```
1322     print('Since he despises the soldier in charge
     of the dining hall because he bullies everyone the
     guard '
1323           'decides to share vital information about
      him, the twist being the higher the score you get
     the more'
1324           ' information he dispels')
1325     print('If you get above the score of 75 u will
     get the key to the broad cast room')
1326     time.sleep(5)
1327     time_limit = 60
1328     if user1.skill < 0:
1329         time_limit = 45
1330     elif user1.skill > 2:
1331         time_limit = 75
1332     score = aimTrainer(time_limit)
1333     print('Your score is:', score)
1334     time.sleep(1)
1335     if score == 69:
1336         print('[Guard Gorbachev]: Nice')
1337         print('[Guard Gorbachev]: Private Sawcon
     always has the key to the broad cast room in his
     pocket')
1338         user1.skill += 2
1339     elif score > 100:
1340         print('[Guard Gorbachev]: Officer you are
     cracked')
1341         print('[Guard Gorbachev]: Private Sawcon is
      really obsessed with his body, if you want to woo
     him mention his '
1342               'body')
1343         print('[Guard Gorbachev]: Private Sawcon
     has always been scared of guns')
1344         print('Here is your promised key')
1345         if not Room_Key in user1.inventory:
1346             user1.change_inventory(Room_Key, True)
1347         user1.skill += 3
1348     elif score > 75:
1349         print('[Guard Gorbachev]: You are quite the
      sharpshooter, but always remember switching to
     your pistol is '
```

```
1350                'faster than reloading')
1351          print('[Guard Gorbachev]: Private Sawcon
     has always been scared of guns')
1352          print('Here is your promised key')
1353          user1.change_inventory(Room_Key, True)
1354          user1.skill += 2
1355      elif score > 60:
1356          print('[Guard Gorbachev]: Well done officer
     , looks like you are still in form to go the
     battlefield')
1357          print('[Guard Gorbachev]: Private Sawcon
     always has the key to the broad cast room in his
     pocket')
1358          user1.skill += 1
1359
1360      else:
1361          print('[Guard Gorbachev]: Officer your
     performance has been subpar to the standards I
     recommend coming back in'
1362                ' a few hours, I have no information
     to give you due to such abysmal performance')
1363      shooting_range.change_room_status()
1364      Add_Value_to_Data(user_username, 'Shooting
     range score', score)
1365      room_transition(generals_room, None,
     shooting_range, Generals_Room, None)
1366
1367
1368 def hitler_maze_run():
1369      # Constants to be set
1370      SCREEN_WIDTH = 1000
1371      SCREEN_HEIGHT = 800
1372      wall_image = pygame.image.load('pygame2.jpg')
1373      square_image = pygame.image.load('white_square.
     jpg')
1374      empty_image = pygame.image.load('black_square.
     jpg')
1375      dot_image = pygame.image.load('dot.jpg')
1376      clock = pygame.time.Clock()  # To set the frame
      rate
1377      SCREEN = pygame.display.set_mode((SCREEN_WIDTH
```

```
1377 , SCREEN_HEIGHT))
1378
1379     def create(_maze, path):
1380         clock.tick(120)
1381         SCREEN.fill((0, 0, 0))
1382         m2 = _maze[:]  # creates a copy
1383
1384         for item in path:
1385             m2[item[0]][item[1]] = '.'
1386
1387         m2[path[-1][0]][path[-1][1]] = 'M'
1388
1389         for y in range(len(_maze)):
1390             row = _maze[y]
1391             for x in range(len(row)):
1392                 item = row[x]
1393                 if item == '1':
1394                     SCREEN.blit(wall_image, (46 * x
     , 45 * y))
1395                 elif item == 'M':
1396                     SCREEN.blit(square_image, (46
      * x, 45 * y))
1397                 elif item == '.':
1398                     SCREEN.blit(dot_image, ((46 * x
     ) + 23, (45 * y) + 22))
1399                 elif item == '2':
1400                     SCREEN.blit(empty_image, (46 *
     x, 45 * y))
1401         pygame.display.update()
1402
1403         for event in pygame.event.get():
1404             if event.type == pygame.QUIT:
1405                 pygame.quit()
1406
1407     def get_maze(file):
1408         f = open(file, 'r')
1409         reader = csv.reader(f)
1410         maze = []
1411         for line in reader:
1412             maze.append(line)
1413         return maze
```

```
1414
1415        def navigate(path):
1416            time.sleep(0.3)
1417            clock.tick(120)
1418            cur = path[-1]
1419            create(maze, path)
1420            poss = [(cur[0], cur[1] + 1), (cur[0], cur[
        1] - 1), (cur[0] + 1, cur[1]), (cur[0] - 1, cur[1
        ])]
1421            choice = randint(0, 2)
1422            # Randomly chooses desired path of
        traversal
1423            if choice == 1:
1424                # random traversal
1425                random.shuffle(poss)
1426            elif choice == 2:
1427                # breadth first tree traversal
1428                poss = [(cur[0] + 1, cur[1]), (cur[0
        ] - 1, cur[1]), (cur[0], cur[1] + 1), (cur[0], cur[
        1] - 1)]
1429            elif choice == 3:
1430                # depth first tree traversal
1431                poss = [(cur[0], cur[1] + 1), (cur[0],
        cur[1] - 1), (cur[0] + 1, cur[1]), (cur[0] - 1, cur
        [1])]
1432
1433            # This is the recursive algorithm that call
         itself till the end of the maze is reached
1434            for item in poss:
1435                # to keep in check so that algorithm
        doesn't go off the maze
1436                if item[0] < 0 or item[1] < 0 or item[0
        ] > len(maze) or item[1] > len(maze[0]):
1437                    continue
1438                # to avoid colliding into walls
1439                elif maze[item[0]][item[1]] in ['1', '2
        ']:
1440                    continue
1441                # to avoid retracing the path
1442                elif item in path:
1443                    continue
```

```
1444             # if user reaches end of the maze
1445             elif maze[item[0]][item[1]] == 'B':
1446                 path = path + (item,)
1447                 create(maze, path)
1448                 input('Hitler enters the Mysterious
      Room and you hear a gunshot, you realize the rein
     of'
1449                       ' the Third Reich is all but
     over\nPress Enter to finish the Game\n>')
1450                 exit()
1451                 pygame.quit()
1452 
1453 
1454             else:
1455                 new_path = path + (item,)
1456                 # once the path is updated the
     algorithm calls upon itself
1457                 navigate(new_path)
1458                 maze[item[0]][item[1]] = '2'
1459                 create(maze, path)
1460                 time.sleep(0.3)
1461                 pygame.display.update()
1462 
1463     maze = get_maze('maze.csv')
1464     run = True
1465     start = ((1, 0),)
1466     while run:
1467         navigate(start)
1468         for event in pygame.event.get():
1469             if event.type == pygame.QUIT:
1470                 run = False
1471                 pygame.quit()
1472 
1473 
1474 user_username = login_system()
1475 user_setup()
1476 intro()
1477 Main__Hall()
1478 
```

Testing

The link to the testing of the game is here:

https://youtu.be/kbvktbtbrFc

Evaluation

Recap of User and Game objectives:

User needs and objectives:

1) General:

1) All the files needed for the program should be provided with it
2) Game should be able to run on command prompt for any os system
3) User should be able to understand what is expected of them for input
4) Game should run without any bugs or errors of sorts

2) Story Progression:

1) Story should be easy to follow and then user should be able to navigate the rooms easily
2) The user should be able to travel to other rooms and engage with the story but still be prompted to complete the storyline in order
3) The user should be able to interact not only with the NPCs set for each room but with the room itself
4) The user should be rewarded for specific answers that change the due course of their storyline and should be notified about it

3) Minigames:

1) The minigames should load immediately when called and should close either when the game has finished or if the user decided to quit early
2) If the user decides to quit early the game should still return a score and not crash
3) The maze game should be able to use a recursive algorithm to give the user an opponent
4) The algorithm should follow either breadth first, depth first or a random graph traversal method depending on the users' attributes
5) Use the time library to keep track of time for mini games with a time limit

4) Room and Characters:

1) The use of inheritance to efficiently classify the main boss from his subordinates as this would allow the subordinates to have attributes such as loyalty which can be attested and changed but has no relevance to the main boss
2) Linking the room class with the item class using linked list operations to allow items to be randomly assigned to a location in the room inventory and allow the user to search the room to find it
3) Create a dictionary that allows rooms to be linked together using directions as keys, enabling the user to travel from one room to another easily
4) The use of property setters to override any mis-inputs of data
5) The use of aggregate and composition to enable the program to pass objects as parameters into different class methods to manipulate specific attributes of the desired object
6) Use class attributes to ensure once the room is visited once the subroutine is not played again to maintain data integrity

5) Login System:

1) Different logins to secure the data saved for a particular user so that multiple users could play on the same system and compare their outcomes
2) Save data to an external file out of the main program to ensure safekeeping once the game has finished
3) Encrypting the password and essential data in the file using hashing methods to enable privacy and security
4) Ensure only users with the right login credentials can access the data

6) User Menu:

1) The user should be provided with a multitude of options to choose from at the end of each subroutine/ room
2) These options should use complex user defined Object-oriented programming models such as composition and aggregation in order to enable the user to directly interact with character and room inventories
3) The appending and removal of items from user and character inventories should be implemented using complex list operations storing objects
4) The user should get a limited number of tries to access the menu and should be able to exit it whenever needed

7) Save/Load Menu:

1) Menu to navigate different levels and different difficulties of the story mode:
   Upon finishing each level or era of history the user would be able to play that level again being able to overwrite their current data and change their decisions affecting the finality of the game. Each decision would be stored in a database and the user would be able to overwrite that data with a simple click of a button. This would be done using the file handling methods that come built in python such as open. The overwriting of data could be implemented using stack operations to return to latest saved stage of the game
2) Graph nodes and tree traversal should be used to notify the branch of the decision tree the user has reached in the program
3) Erase data from a user profile to start a new game
4) Check all the easter eggs/ artefacts they have collected:
   A list of all the artefacts collected by the user and stored on a database would be listed
5) Have an option to pause the game and decide to save the data, reset from their last checkpoint, reset the level or quit to main menu
6) Use a merge sort or a bubble sort algorithm to determine the exact point of entry to load the user into the game with the previously input data intact

Objectives check box:

1) General Objectives

Objective Number	Is it met?	Reason if met/not met/partially met
1.1	Yes	The objective is fully met as the files necessary for the code to be executed are provided with the code as well
1.2	Yes	The objective is fully met as the code can only run on the terminal as running it on an IDE would result in bugs as the getpass module and screen refresh subroutine aren't registered properly on an IDE terminal (PyCharm)
1.3	Yes	The objective is fully met as the user is given hints throughout the game if read properly and the shooting range is always open for them to receive the key to proceed forward
1.4	Yes	The game is tested a lot of times before finalizing and in the test run video it is shown, 3 different inputs (normal, extreme and invalid) of data are given and the program responds accordingly to all 3

The general user objectives are fully met as the code can run smoothly without any hiccups

2) Story Progression Objectives

Objective Number	Is it met?	Reason if met/not met/partially met
2.1	Yes	The objective is fully met as the user is guided to other rooms through the use of prompts from other NPCs in the game
2.2	Yes	The objective is fully met as the game only progresses forward by restricting access to certain rooms until the user has visited the room needed
2.3	Partially	This objective is only partially met as sure the user gets to search 2 different rooms using the menu options but that is the only interaction they have with the room and have no other accessibility to it such as for example exploring the room
2.4	Partially	This objective is partially met due to the fact that the user does get rewarded for certain comments such as the conversion with General Aladeen but then are not notified about it and only find out when their statistics are displayed

The Story Progression objectives are almost fully met and just need a small tweak in regards of providing the user with more options and information in regards of visibility

3) Minigames Objectives:

Objective Number	Is it met?	Reason if met/not met/partially met
3.1	Partially	This objective is only partially met as even though both the minigames close upon reaching the end or if the user closes it, the final maze program has a small bug where it can close if the user forces it to but once it reaches the end of the maze it crashes
3.2	Yes	The objective is fully met as the maze does not return a score of any sort and thus it crashing or bugging has a less overall impact on the whole program
3.3	Partially	This objective is only partially met as a recursive algorithm is implemented however it is not to pot ray the opponent but rather to display a fun little ending for the user
3.4	Partially	This objective is only partially met as even though the choice to traverse between 3 different traversal methods is given in reality every time the item navigates the maze it can re choose a new path of traversal so i.e., the maze will not be travelled only using one method but rather a mix of all 3

3.5	Yes	The objective is fully met as the time module is used to keep the track of time in the background during the Aim Trainer minigame and the game closes as soon as the timer hits 0

The minigame objectives were well met with achieving most results which were necessary however certain techniques such as knowing which system exit to use when could be implemented to achieve the smooth running of the game.

4) Room and Characters Objectives:

Objective Number	Is it met?	Reason if met/not met/partially met
4.1	Yes	The objective is fully met as the class Hitler inherits all attributes and methods from the Character class while creating methods of its own to be used. However, this might not be the most efficient implementation as a lot of methods and attributes are not needed by the class and thus are wasted such as those highlighted in the objective
4.2	Yes	The objective is fully met as the methods search room and item placement help hide the location of the item in the room by exchanging returns into each other's parameters
4.3	Yes	The objective is fully met as the linking of rooms is implemented using a dictionary. The direction is given as the field and the object (the room) is given as the value. This way the room link method allows any 2 rooms to be linked to each other in any way possible
4.4	Yes	The objective is fully met using the implementation of property declarations which act as getters and setters but in a more secure and efficient way
4.5	Yes	The objective is fully met as aggregation and composition are used in different scenarios to help implement different designs, such as composition is used to create a team for the combat while aggregation is used for placing an item in the room
4.6	Yes	The objective is fully met as the 'room status' attribute ensures the validity by returning a Boolean value that confirms whether or not the user has entered the room

The room and character objectives were all met flawlessly however in hindsight it was a better design to implement a whole new class for Hitler such as Class Boss Battle and have the Hitler class inherit from there instead as this would provide it with the necessary attributes and methods required for a boss battle instead of extra ones, so over here the objective should be modified

5) Login System Objectives:

Objective Number	Is it met?	Reason if met/not met/partially met
5.1	Partially	This objective is only partially met as different logins from various users is allowed through the use of usernames and passwords however a system to compare statistics like on a leaderboard has not been implemented yet and is a potential next step for the program
5.2	Yes	The objective is fully met as all of the data is stored on a Json file separate from program running on the python file. This executed using the help of the Json library
5.3	Partially	This objective is only partially met as encryption methods have been used in the form of hashlib library however it was a user expectation to create a more secure system of encryption possibly trying to create a custom hashing algorithm
5.4	Yes	The objective is fully met as due to the use of get pass and passwords in general, during the login phase it is impossible to guess another user's password as it could be as short as 1 letter or a long paragraph, thus only users that remember their passwords could access their data

The Login system objective was one of the most difficult to implement but worked out the best as not only did the getpass module give a more authentic feel to the running of the section but the option to login anytime and view your details. The only changes needed to this section of the program would be an additional build up to exist features such as creating a leaderboard and creating a custom hashing algorithm

6) User Menu Objectives:

Objective Number	Is it met?	Reason if met/not met/partially met
6.1	Yes	The objective is fully met as the user is given 5 different options as to what to do after finishing the subroutine planned for the room
6.2	Yes	The objective is fully met as when implementing the menu system for each room, the room and the main character associated with the room are passed into the menu as objects for the methods to be called upon so the user can interact with them
6.3	Yes	The objective is fully met as it is hard to compare string type inputs from the user to objects thus separate lists are implemented to append items and then compare the objects' name attributes with the input as both are the same type of data

6.4	Yes	The objective is fully met as the maximum number of uses of the menu is 5 whether the user spends it on one option 5 times or tries out all of the 5 options given. The 6th option also allows the user to leave the menu promptly and continue with the game if they are in a hurry

The user menu objectives are fulfilled in a very simplistic yet efficient manner, where passing just 2 objects per room takes care of the whole process and is easier to debug for errors, and even easier to add more menu options to. However, in hindsight it would have been better to tell the user they have only 5 tries at the menu and better yet show the counter after every option

7) Save/Load Menu Objectives:

Objective Number	Is it met?	Reason if met/not met/partially met
7.1	No	This objective has not been met at all as there is an option to overwrite current data using file handling on append mode but the implementation of saving decisions as data using stack operations is yet to be implemented
7.2	No	This objective has not been met at all as there are graphs or nodes implemented throughout the program. As such it's not possible to use any sort of traversal method to come to the decision of the user
7.3	Partially	This objective is only partially met as though there is no option to erase or delete your data, using another username or just simply manually deleting the data of the JSON file can secure the objective. However, a feature to erase existing data is needed, perhaps presented as an option to the user once they have logged in using the login system
7.4	No	This objective has not been met at all as all of the items collected at the end of the game are either used in defeating Hitler or not recorded
7.5	Partially	This objective is only partially met as though the data is saved by the program automatically as certain crucial points in the game for example after the completion of a minigame, this still does not mean the user's major decisions are saved and they are reliant on reaching till a very crucial checkpoint for the game to save the data. This saving feature is not even mentioned to the user thus they do not even know when their data is getting saved
7.6	No	This objective has not been met at all as not only there is no graph for any sort of algorithm to be implemented but the only point of entry to the game is login menu, i.e., the very start of the game

The save/load objectives are the objectives that were missed out the most as the objectives are either only barely partially met or not met at all. Moving forward in the game the implementation of a load system should meet most if not all the objectives in this box and this load system could implemented using a class tree with right, left and center nodes for each branch of decision in the game, using this graph various traversal methods could then be implemented

End User rating:

After introducing the game to a few friends, the overall reaction to the game was a very positive experience including a few constructive remarks to make the game better which are included in the section below. Most found the game to be really fun and entertaining for a text-based adventure game as not only was it historically accurate and had little easter eggs for everyone to enjoy but it also had small minigames that entertained the students that weren't particularly interested in history.

An example of this would be students trying to get the highest score at the shooting range to compare with each other or conserve the most lives at the brick game. The icing on the cake for most was the small ending given to everyone once they defeat Hitler as many were caught off-guard by what it was and represents.

I conducted an interview with the 2 potential users mentioned in my analysis and their improvements are marked below as well. Their remarks were mostly positive as they were surprised by the number of features in a small text-based adventure game.

Conclusion (changes to the game going forward):

Most of the changes required to this game are changes done to the save/load menu as data is surely saved but as of now there is no way to access that data, and even the data that is saved is quite elemental and does not have much importance. The possible solutions to this menu set up is already mentioned above with the use of trees, nodes and traversals

However there were minor details that could help change the game such as the repetition of commands or negative remarks for the failure of the task, for example the failure to retrieve an item after plundering the room, through the recommendation of a user a possible solution to this would be creating a list of different remarks and make the program randomly choose from it, even though this is a tiny implementation it made the user feel like the game would feel less repetitive and more interesting

Another feature recommended was an edit to the aim training minigame. The change was to change the size of the balls more randomly and make them even smaller and harder to hit as you progressively get more points. This sort of system could be implemented using a feedback loop that checks the score and changes the dots variables and limitations accordingly

The features suggested by my friends (the end potential users) in the interview were to focus more on the graphics and the audio. This could be done using other python modules with better routine handling such as the asyncio module – Asynchronous Input/Output module which is in charge of handling 2 different events processing independent of each other. These visual and audio graphics however could be better implemented using a more advanced game engine too.

The final feature to be added if I had more time was the option to view your decision tree, explaining how you ended up in a situation and what were your other options and then give you the option to replay that chapter again to see a different outcome. This feature falls heavily in theme with the game as the entire point of history is to teach you from your previous mistakes.

The link to my interview with the end user, which I used to draw up my evaluation:

https://www.youtube.com/watch?v=ELezYVVaWCc