Task completed:

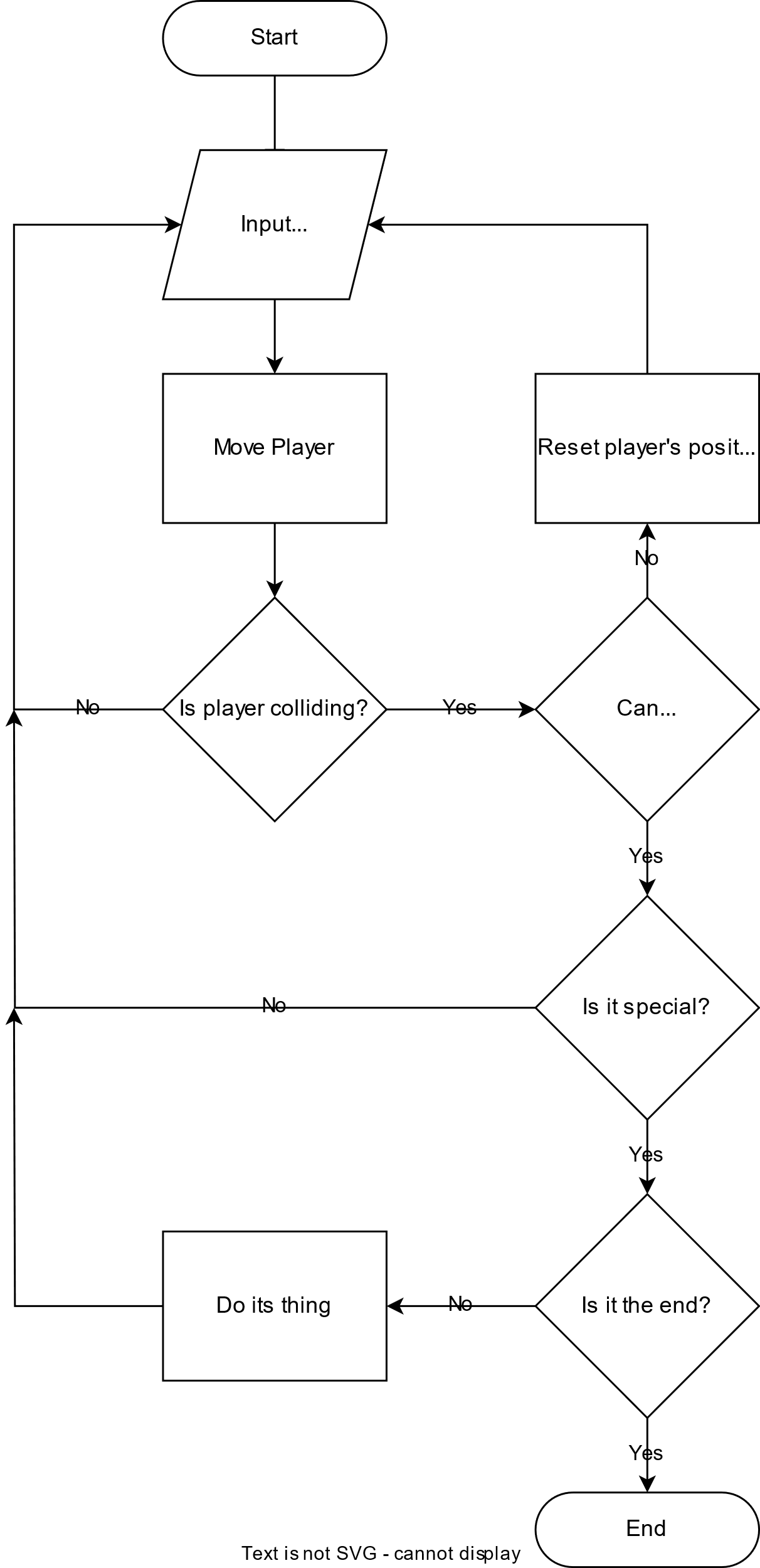
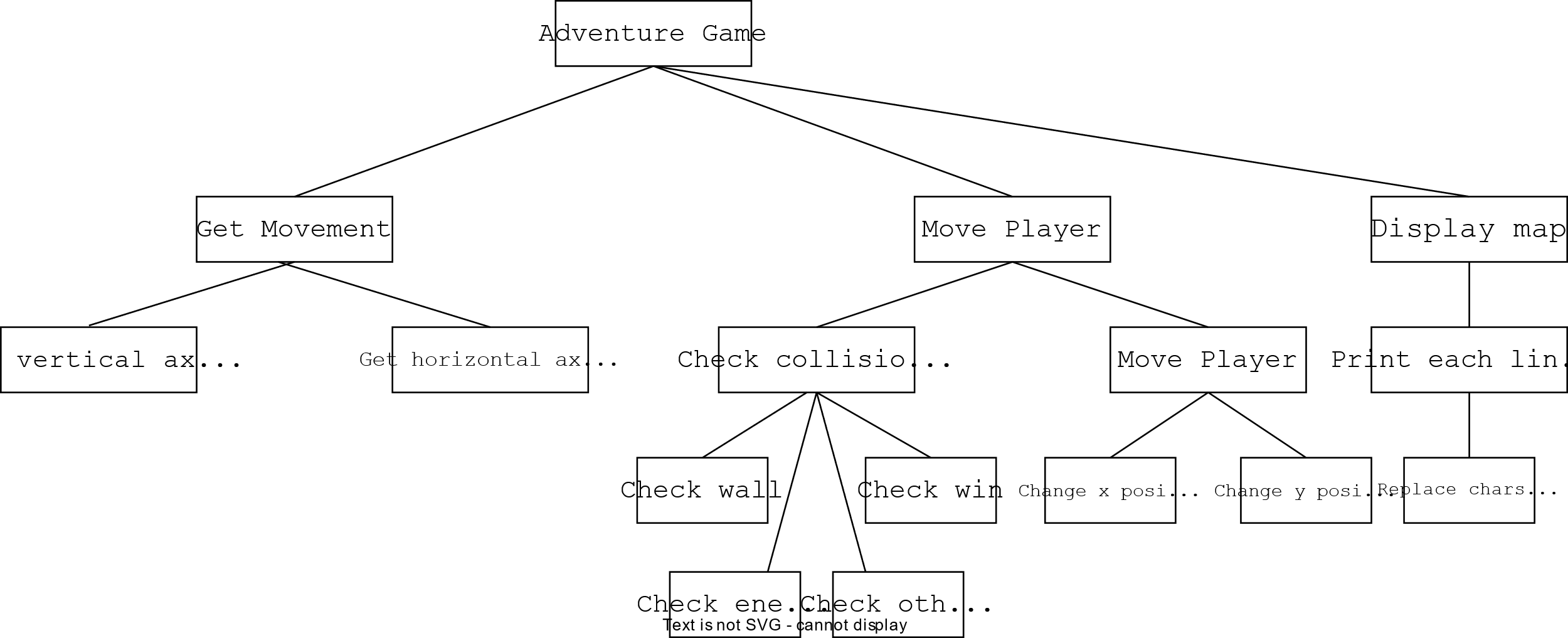
| Date started | Date completed |
| --- | --- |
| idk | N/A |

Analysis

Try and create 3 or more key success criteria for your program.

Success Criteria:

1. Movement works
2. Chests work
3. Authentication works



Design

* *You may like to create a flow charts which will show broadly how your program will work. If so include your flow chart in this section.*

Test design

* *Think of tests that you can carry out to see if your system works*
* *Remember to try and use normal, boundary and erroneous tests.*
* *If you wish to, you may add more tests to the table.*

My tests:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | What am I testing? | What data will I use? | Normal/Boundary/Erroneous? | Expected Result |
| 1 | Login | nathan; password | normal | \*let user play game” |
| 2 | Login | this\_is\_wrong: password | erroneous | \*doesn’t let user play game\* |
| 3 | Movement | \*move player into wall\* | N/A | \*player doesn’t move into wall\* |
| 4 | Lock and Key | \*move player into lock, key then lock again\* | N/A | You need a key!  \*Player doesn’t move onto lock\*  \*Player gets key\*  \*Player does move onto lock\* |
| 5 | End | \*move player to end square\* | N/A | \*Game ends\* |
| 6 | Chests | \*move player onto chest\* | N/A | \*Player gains loot\* |

Development

* *Copy and paste your code into this section*
* *Remember to try and add comments to your code to make it more readable!*

My program code:

# -\*- coding: utf-8 -\*-

import sys

from typing import Callable

import keyboard

import os

from classes.auth import Auth

from classes.inventory import Inventory, LootTable

class Game:

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

self.player\_start\_positon = [2, 2] # ? x, y (0 indexed)

self.player\_positon = self.player\_start\_positon

self.current\_map = maps["1"]

self.running = True

self.end\_message = ""

self.action = False

self.inventory = Inventory()

self.bonus\_message = ""

self.chest\_loot\_table = LootTable(

{

"gold": (

(1, 3), 1

),

"key": (

(1, 1), 1

)

}

)

def write\_error(self, function: Callable, error: Exception) -> None:

if not os.path.exists("errors.log"):

a = open("errors.log", "w")

a.close()

with open("errors.log", "a") as f:

f.write(f"{function}: {error}\n")

def display\_map(self, clear: bool = True) -> None:

if clear:

self.clear\_lines(len(self.current\_map["map"]) + 2)

else:

print("\n"\*3)

try:

self.main\_output = ""

for y\_index, line in enumerate(

self.current\_map["map"]

): # \* Goes through each horizontal line in the map

self.output = ""

for x\_index, char in enumerate(

line

): # \* Goes through each character in the line

if (

x\_index == self.player\_positon[0] and y\_index == self.player\_positon[1]

): # \* Checks for the players position being the current position

self.output += "@"

else:

self.output += char

self.main\_output += self.output + "\n"

self.main\_output = (

self.main\_output.strip()

) # \* Removes the extra \n at the end of the output

except Exception as e:

self.write\_error(self.display\_map, e)

finally:

if do\_fancy\_tiles:

for key, tile in fancy\_tiles.items():

self.main\_output = self.main\_output.replace(key, tile)

self.inventory\_output = f"Inventory: {self.inventory}"

print(self.bonus\_message)

print(self.inventory\_output)

print(self.main\_output)

self.bonus\_message = ""

def replace\_char\_at\_index(self, text: str, replace: str, index: int) -> str:

self.to\_return = list(text)

self.to\_return[index] = replace

return("".join(self.to\_return))

def clear\_lines(self, count: int = 1) -> None:

for \_ in range(count):

sys.stdout.write("\033[F")

sys.stdout.write("\033[K")

def move\_player(self, x: int, y: int) -> None:

# print(f"Player moving {x},{y}") #! DEBUG

self.player\_vector = [x, y]

self.old\_player\_positon = self.player\_positon.copy()

# \* Adds the player\_vector to the current player position

self.player\_positon[0] += self.player\_vector[0]

self.player\_positon[1] += self.player\_vector[1]

# \* Checks if the players next position is a wall

try:

self.current\_tile = self.current\_map["map"][self.player\_positon[1]][self.player\_positon[0]]

if self.current\_tile == "#": # ? Wall tile

# \* Resets the players position

self.player\_positon = self.old\_player\_positon

if self.current\_tile == "~": # ? End tile

self.end\_message = "You won!"

self.finish()

if self.current\_tile == "+": # ? Key tile

self.bonus\_message = "You got a key!"

self.inventory.add\_item("key", 1)

self.current\_map["map"][self.player\_positon[1]] = self.replace\_char\_at\_index(self.current\_map["map"][self.player\_positon[1]], " ", self.player\_positon[0]) #\* Replaces the key's location with a blank character

if self.current\_tile == "=": # ? Lock tile

if self.inventory.use\_item("key", 1):

self.current\_map["map"][self.player\_positon[1]] = self.replace\_char\_at\_index(self.current\_map["map"][self.player\_positon[1]], " ", self.player\_positon[0]) #\* Replaces the lock's location with a blank character

else:

self.bonus\_message = "You need a key"

self.player\_positon = self.old\_player\_positon # \* Roll back players position

if self.current\_tile == "\*": # ? Chest tile

self.current\_map["map"][self.player\_positon[1]] = self.replace\_char\_at\_index(self.current\_map["map"][self.player\_positon[1]], " ", self.player\_positon[0])

self.inventory.gain\_loot(self.chest\_loot\_table, 1)

self.bonus\_message = "You opened a chest!"

self.door\_check()

self.action = True

except Exception as e: # \* Checks if the player is off the map

self.write\_error(self.move\_player, f"{e.\_\_context\_\_} : {e}")

self.player\_positon = self.old\_player\_positon

def door\_check(self):

for door\_position, door\_data in self.current\_map["doors"].items(): #\* Iterates through all doors in the current room

if tuple(self.player\_positon) == door\_position:

self.player\_positon = door\_data[2].copy() #? New player position

self.current\_map = maps[door\_data[1]].copy()

def finish(self):

self.running = False

if "-t" not in sys.argv:

auth = Auth("./login.json")

user = auth.check\_user(input("Enter your username\n>> "), input("Enter your password\n>> "))

if not user:

quit()

bonus\_message = ""

default\_tile = "\u25CD"

fancy\_tiles = {

"#": "\u25FC", # ? Wall

"@": "\u26F6", # ? Player

"~": "\u25FB", # ? End tile

"v": "\u23F7", # ? Down arrow

"^": "\u23F6", # ? Up arrow

">": "\u23F5", # ? Right arrow

"<": "\u23F4", # ? Left arrow

"+": "🗝", # ? Key

"=": default\_tile, # ? Lock

"\*": default\_tile, # ? Chest

}

do\_fancy\_tiles = False

map\_1 = [

"##########", # ? 0,0 -> 9,0

"# ## >",

"## ###",

"##########", # ? 0,3 -> 9,3

]

map\_2 = [

"#######", # ? 0,0 -> 7,0

"< #~#",

"## =#",

"##v####", # ? 0,3 -> 7,3

]

map\_3 = [

"##∧##", # ? 0,0 -> 4,0

"#\* ##",

"## +#",

"#####" # ? 0,3 -> 4,3

]

maps = {

"1": {

"map": map\_1,

"doors": {

(9, 1): ["right", "2", [1, 1]], #? door\_pos: [door\_direction, new\_map, new\_player\_pos]

}

},

"2": {

"map": map\_2,

"doors": {

(0, 1): ["left", "1", [8, 1]],

(2, 3): ["down", "3", [2, 1]]

},

"from": {

"left": [1, 1],

"down": [2, 2]

},

},

"3": {

"map": map\_3,

"doors": {

(2, 0): ["up", "2", [2, 2]]

}

}

}

if "-t" not in sys.argv:

option = input("Fancy tiles? y/n\n>> ")

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

do\_fancy\_tiles = True

option = input("Erase each time? y/n\n>> ")

do\_clear = option.lower() == "y"

else:

option = "n"

do\_clear = True

game = Game(maps)

print("\n" \* len(game.current\_map["map"]))

keyboard.add\_hotkey("w", game.move\_player, (0, -1))

keyboard.add\_hotkey("s", game.move\_player, (0, 1))

keyboard.add\_hotkey("a", game.move\_player, (-1, 0))

keyboard.add\_hotkey("d", game.move\_player, (1, 0))

keyboard.add\_hotkey("esc", game.finish)

while game.running:

game.display\_map(do\_clear)

while not game.action:

pass

game.action = False

print(game.end\_message)

input("Here's all of your moves!\n (Press enter to continue and exit)\n")

A screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

Evaluation

* Evaluate how successful your program was. You should like your evaluation to your testing results.
* You should reflect on any new skills you have developed

This section should be approximately 200-500 words.

How successful was my program?

My program achieved all of its goals in a concise and efficient manner.

The program is a text-based adventure game with wasd controls which can be run in the terminal. It has an authorisation system as well as a file reading. When the program is run, it asks the user a series of questions as to how the program should run, such as what textures to use and whether they want the previous level to be erased before showing the new one.

There was slight difficulties when trying to implement doors, but I discovered the bug and removed it after some basic troubleshooting.

My flow chart helped as it allowed me to see what my code needed to do next when the codebase had grown. The structure diagram also helped me by allowing me to see how my functions can be broken down and make it so my code is more legible.

During my programming, I learned about more descriptive type annotation as well as improving my skills in object orientated programming. I also used many skills I learned in class as well as from my previous knowledge.