AP CSP Create Task Write-Up

2a)

The programming language I used to code my program is Python. The purpose of my program is to entertain the user through a fun game. The video is sped up to fight time constraints. The video shows the function of the game. The game starts by setting up the maze. Instructions are given out as to how to play the game and how to win. The user then plays the game and tries to collect the 4 keys, while evading the purple enemies. The player, unfortunately, was not able to evade the purple enemies and died.

2b)

I created the game by myself. I started out by designing a simple 10 by 10 block maze. I then created the createmaze method to make a maze. The 10 by 10 maze was very easy to solve, according to my classmates. So, I then redesigned the maze to be 25 by 25 blocks. I then went back in and reconfigured the createmaze method to make a larger maze. I then placed the player and gate. I then created playercontrol by binding the user's arrow keys to methods to move the player up, down, left, and right. The player had to navigate to the gate and the game would end.

My game was still simple. I still had time to make changes, so I then went back and added enemies that the user had to evade. The player now had to get four keys and then he or she would be able to open the gate. Near the end, I had an issue with making the gate only open after four keys were collected. I played around and was able to solve this issue by making the player.key value reach four only after the four keys were collected.

2c)

```
if intro():
    screen.tracer(5)
    createmaze(layout)
    moveallenemies()
    instructions()
    playercontrol()
else:
    print("Alright have a good day")

while True:
    for key in keys_loc:
        if collision(key):
            player.key+=1
            key.materialize()
            keys_loc.remove(key)
    for enemy in enemies_loc:
        if collision(enemy):
            enemy.materialize()
            enemies_loc.remove(enemy)
            player.lives-=1
            print("Lost 1 life.",player.lives,"lives remaining")
    screen.update()
    if player.lives==0:
        print("You have died")
        screen.clear()
        break
    if collision(gate) and player.key==4:
        print("You have won!!! Great Job!!!")
        screen.clear()
        break
```

```
def intro():
    print("Welcome to DungeonEscape!!")
    userpref=int(input("Enter 1 to start game or 0 to quit"))
    if userpref==1:
      return True
    elif userpref==0:
      return False
```

```
def collision(thing):
    delta_xcor=player.xcor()-thing.xcor()
    delta_ycor=player.ycor()-thing.ycor()
    diag_length=math.sqrt((delta_xcor**2)+(delta_ycor**2))
    if diag_length <= 6:
        return True
    else:
        return False
```

The main algorithm starts by using sequencing to start the game. The first sub-algorithm uses the intro method to ask the user if they want to play the game by entering 1 to start and 0 to quit. If the user enters 1, the maze is created, enemies are instructed to move. The instructions and player controls are given to the user. If the user enters 0, a message is printed out and code stops.

The second sub-algorithm uses for loops and if statements. The collision(112-119) method checks if the player has run into a key, enemy, or gate. The collision method uses the Pythagorean theorem to find the distance between the player and the object put in. If the distance is 6 or lower, True is given out. Else, false is returned. Once the player.lives reaches 0, the game ends. If the player has 4 keys and collides with the gate, the player wins. The collision method is repeated for every value in the enemies_loc and keys_loc lists using for loops. This algorithm runs the entire game from start to finish. This algorithm checks if the player has won or lost too.

2d)

```
6  class Key(turtle.Turtle):
7      def __init__(self,x,y):
8          turtle.Turtle.__init__(self)
9          self.penup()
10         self.goto(x,y)
11         self.speed(0)
12         self.color("green")
13
14     def materialize(self):
15         self.hideturtle()
16         self.goto(2500,2500)
```

```
class Enemy(turtle.Turtle):
    def __init__(self,x,y):
        turtle.Turtle.__init__(self)
        self.speed(0)
        self.penup()
        self.goto(x,y)
        self.shape("triangle")
        self.color("purple")

    def move(self):
        self.direction=random.randint(1,4)
        if self.direction == 1:
            movex=self.xcor()
            movey=self.ycor()+24
        elif self.direction == 2:
            movex=self.xcor()
            movey=self.ycor()-24
        elif self.direction == 3:
            movex=self.xcor()-24
            movey=self.ycor()
        elif self.direction == 4:
            movex=self.xcor()+24
            movey=self.ycor()
        if (movex,movey) not in walls_loc:
            self.goto(movex,movey)
        else:
            self.direction=random.randint(1,4)
        screen.ontimer(self.move,random.randint(100,400))

    def materialize(self):
        self.hideturtle()
        self.goto(-2500,-2500)
```

```
keys_loc=[]
enemies_loc=[]
walls_loc=[]
```

```
72  def moveup():
73      movex=player.xcor()
74      movey=player.ycor()+24
75      if (movex,movey) not in walls_loc:
76          player.goto(movex,movey)
77
78  def movedown():
79      movex=player.xcor()
80      movey=player.ycor()-24
81      if (movex,movey) not in walls_loc:
82          player.goto(movex,movey)
83
84  def moveright():
85      movex=player.xcor()+24
86      movey=player.ycor()
87      if (movex,movey) not in walls_loc:
88          player.goto(movex,movey)
89
90  def moveleft():
91      movex=player.xcor()-24
92      movey=player.ycor()
93      if (movex,movey) not in walls_loc:
94          player.goto(movex,movey)
```

The method createmaze(53-70) refers to two classes called Key and Enemies. This helped reduce the complexity of my program. I was able to create multiple characters with the same functions and features. In my maze, I have 4 keys and 8 enemies. I didn't have to create 12 different characters and waste space. The Enemy class has a move function that moves the character 24 pixels up, down, right, and left randomly. This feature is applied to all instances that the class is used. This allows for all enemies to move randomly.

In 2c, The collision method of the enemies and keys is repeated for each value in the keys_loc and enemies_loc and lists, respectively. These list helps reduce the complexity of the program by holding all the current locations of the walls, keys, and enemies. The enemies list is updated as the enemies move. The walls_loc list helps to make sure that the player only moves within the walls and doesn't go into them. This makes the game harder if the player can't go through walls. This makes the game harder and more enjoyable and logical play.