CSC500: INTRODUCTION TO COMPUTER SCIENCE

DRUNKARD'S WALK

TABLE OF CONTENTS

1. SOFTWARE PROJECT PLAN

1.1. Project Description

In the project, we simulated a random drunkard's walk. The purpose of this project is to calculate the distance that the drunkard traveled after a given number of steps. The drunkard starts out at a random point. At each time unit, the drunkard takes one step in a randomly chosen direction.

1.2. Project Requirements

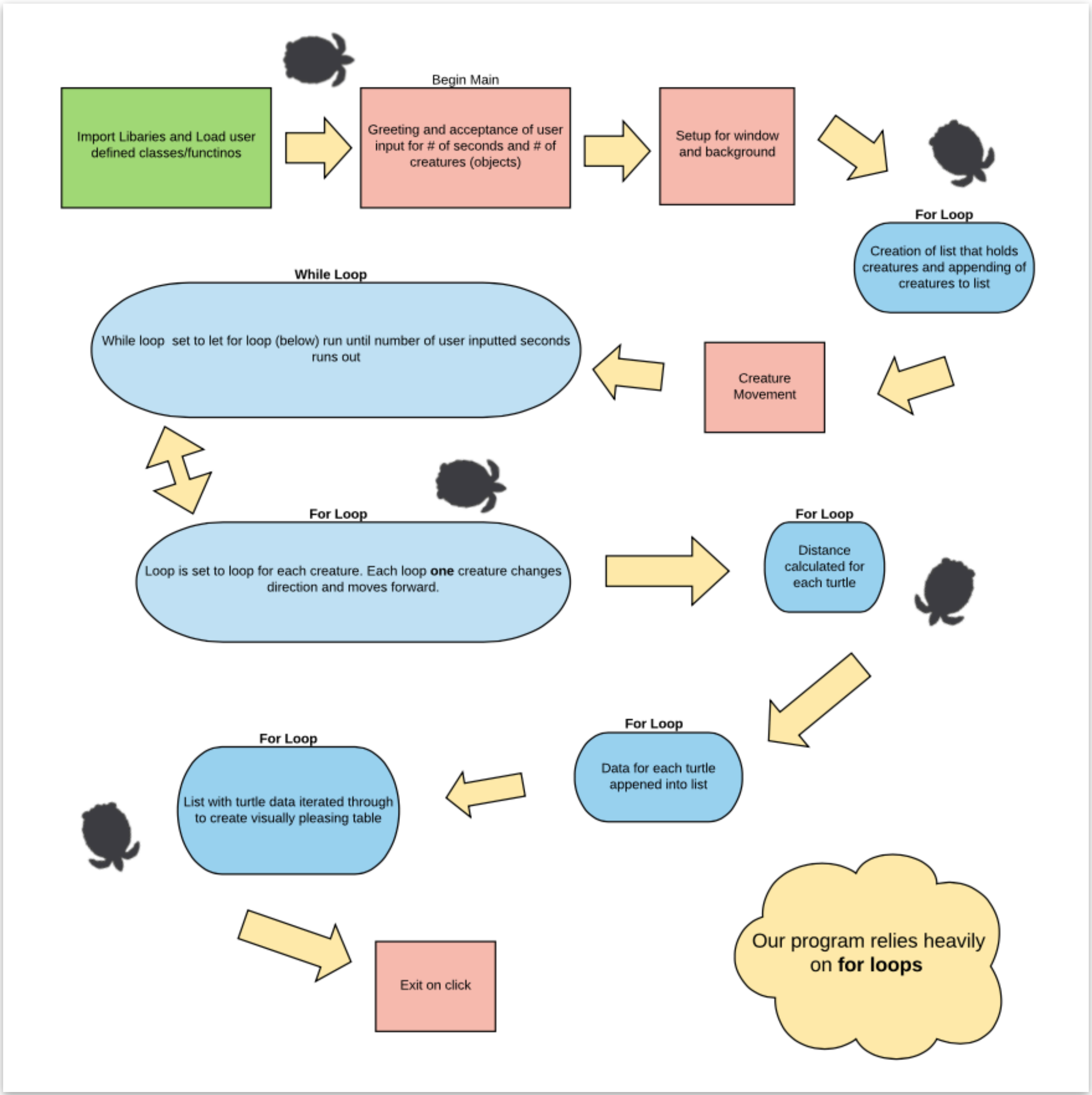
We need a user to enter the amount of input, which is the number of turtles and the number of seconds. We used Microsoft One Drive because we can always access all versions of the project. We are planning to use slack and email for our communications. Clear communication is a very important aspect of our project.

The performance of the project should be smooth and fast as we don't want to wait for our users when the code is running.

Quality of the project should be great in terms of user interface because every person should understand what he or she should do easily when they see the program. Also, it should be testable for any errors that will occur and therefore the code itself should be flexible and should be open to change.

Overall the project should form a complete package with its coding and documentation. Everybody should be able to understand what we did and how it is working.

1.3. Operation Flow Diagram


Import Libaries and Load user defined classes/functinos
Begin Main
Greeting and acceptance of user input for # of seconds and # of creatures (objects)
Setup for window and background
For Loop
Creation of list that holds creatures and appending of creatures to list
Creature Movement
While Loop
While loop set to let for loop (below) run until number of user inputted seconds runs out
For Loop
Loop is set to loop for each creature. Each loop **one** creature changes direction and moves forward.
For Loop
Distance calculated for each turtle
For Loop
Data for each turtle appened into list
For Loop
List with turtle data iterated through to create visually pleasing table
Exit on click
Our program relies heavily on **for loops**

<u>Diagram description:</u>

a. Libraries are imported, functions are defined, and the class is built.

b. The program begins by asking the user how many creatures should be created *(key)* and how many seconds the program should run *(seconds).*

c. The code to setup window.

d. An empty list to hold the creatures is defined. A **for loop** that loops *key* times uses the class to create and append creatures (creatures initialized at random locations, defined by the class).

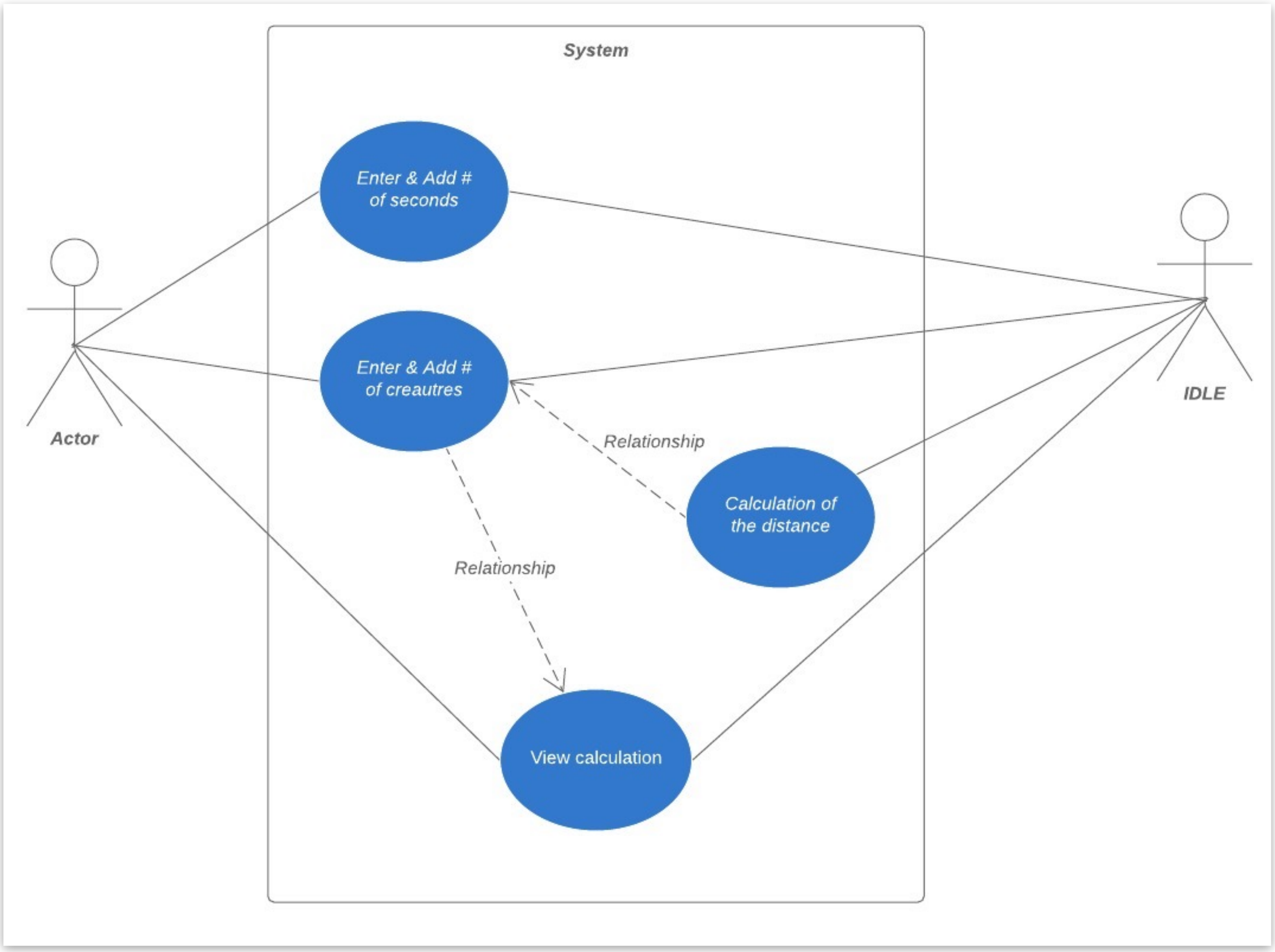
e. Creature movement is regulated by a **while loop** and a nested **for loop**. A while loop runs for number of seconds. A for loop uses the *key* as number of loops. The inner loop moves each turtle, one at a time.

f. A **for loop** is used, with a number of creatures as a range, to draw a line from the creature's starting point to the creature's ending point.

g. A list to hold information regarding each creature is created. Information is appended to the list using a **for loop** of number of creatures.

h. A table is printed to present information on each creature.

1.4. Use Case Diagram



<u>Diagram description:</u>

This use case diagram is a graphic depiction of the interactions among the elements of a system. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled. In our use case diagram, we just added the actor and the program (IDLE). Since this is not a big system application; we have just a couple of use cases with which those system objectives include all the requirements. In the first use case, the actor (user) enters the number of seconds to IDLE and then IDLE asks the user to enter the number of creatures. After the user entered all the system requirements, the system shows the output as a calculation of the distance which the creature traveled in the program. It means that from a user's point of view, the system's behavior responds to a request properly.

1.5. Project Team Members

Members of development group:

- Ilya Samokhyalov
- Ward Sylvester
- Evan Kurpiewski
- Naz Yasar
- Ahmet Uygun
- Asli Onturk
- Cagatay Ozdinc

1.6. Project Resources

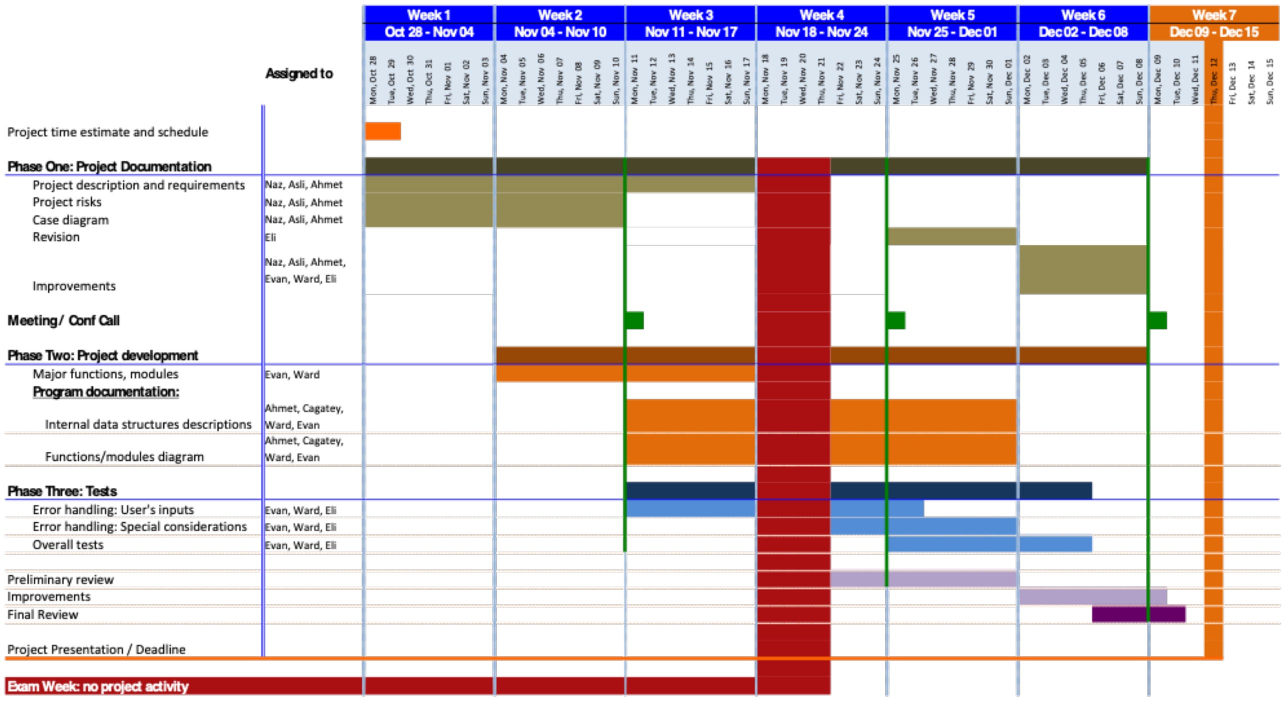
Hardware in use:

- Each member used his or her own personal computer to work on the project and test it.

Software in use:

- Slack: Used to communicate.
- One Drive Cloud: Used to share the documents.
- IDLE for Python: Used to build the project.
- Microsoft Project: Used to schedule and plan the project.

1.7. Time Estimates and Schedule



	Assigned to
Project time estimate and schedule	
Phase One: Project Documentation	
Project description and requirements	Naz, Asli, Ahmet
Project risks	Naz, Asli, Ahmet
Case diagram	Naz, Asli, Ahmet
Revision	Eli
Improvements	Naz, Asli, Ahmet, Evan, Ward, Eli
Meeting/ Conf Call	
Phase Two: Project development	
Major functions, modules	Evan, Ward
Program documentation:	
Internal data structures descriptions	Ahmet, Cagatey, Ward, Evan
Functions/modules diagram	Ahmet, Cagatey, Ward, Evan
Phase Three: Tests	
Error handling: User's inputs	Evan, Ward, Eli
Error handling: Special considerations	Evan, Ward, Eli
Overall tests	Evan, Ward, Eli
Preliminary review	
Improvements	
Final Review	
Project Presentation / Deadline	
Exam Week: no project activity	

1.8. Project Risks

The first risk that we faced coming into the project was group members not knowing their roles. This was quickly taken care of by voting for a team leader, listing project requirements to form roles, and then assign these roles to the various group members. This important piece of the project could have easily caused setbacks but was tackled by the members using their ability to lead, make comprises, and adapt to changing conditions of project requirements.

A second, large risk, that we faced was finding times for all of the group to meet. Although this also could have caused problems, the group did a good job of summarizing meetings for members that could not show up, taking on extra responsibilities, and keeping all the project information in one organized space, OneDrive. The fact that we could easily catch up on what had been decided and what had been accomplished made missed meetings not a problem.

This problem rolled into a third risk, lack of communication. At first, the group used Slack to communicate and to set group meetings. We quickly realized that Slack would not work for this group and decided to shift over to email communication. We are assuming that because the group all checked their emails on a regular basis and found it difficult to add a new form of communication (Slack) to their daily routine, email communication proved to be more efficient.

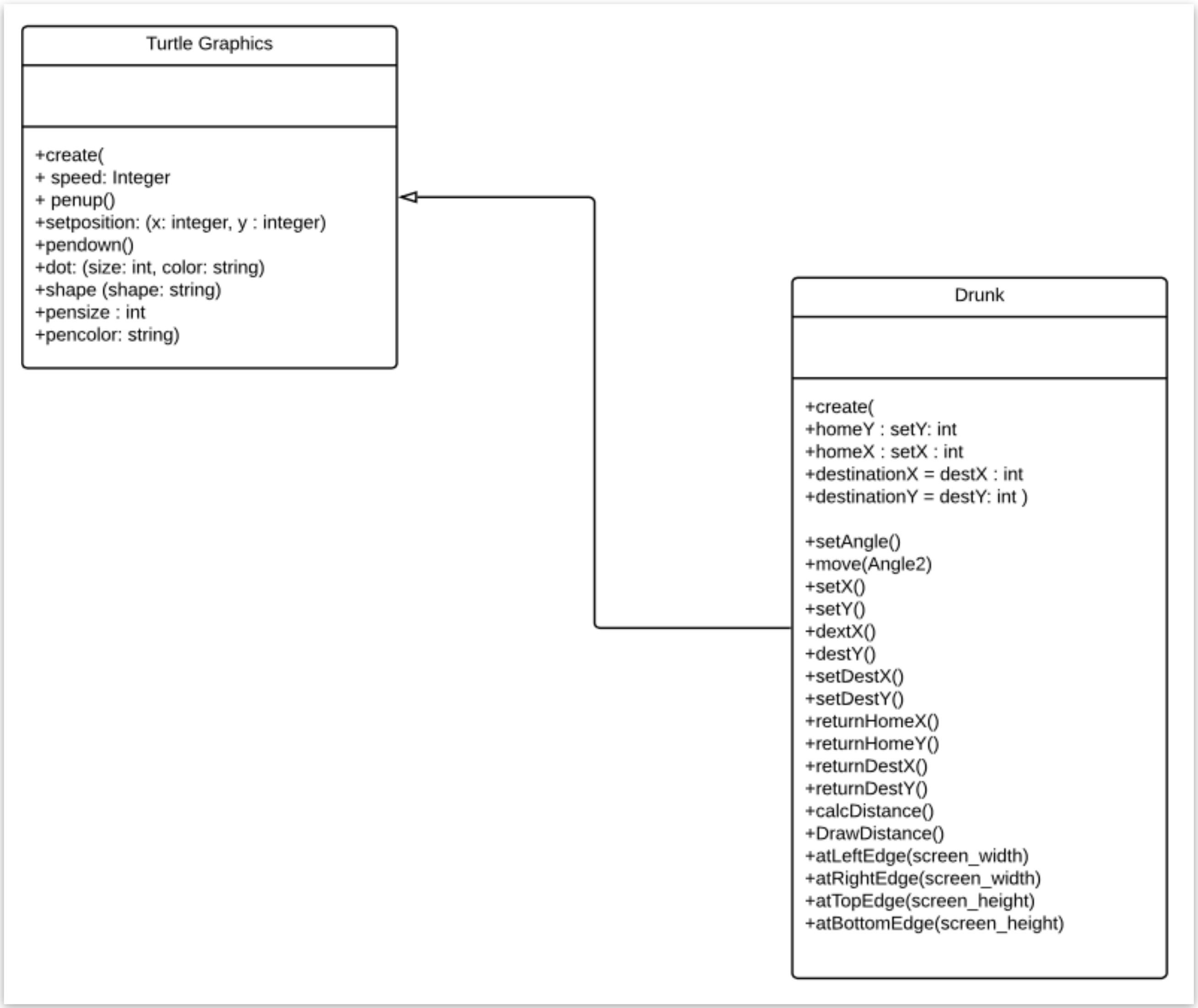
The ability of each group member to voice his or her opinion and be heard by the group made the project run a lot smoother. Although many potential risks could have slowed the project down, the group was able to quickly identify and find solutions to these issues, before they caused too much of a problem.

2. SOFTWARE DESIGN PLAN

2.1. Major Functions and Classes, Methods and Data Structures

The program asks a user to enter a number of seconds he or she would like the program to run. Then it asks to enter a number of turtles the program should generate. After that, a screen pops up and the creatures start their random walks. When all the steps are done, the program calculates the total distance traveled by all creatures and the average distance traveled.

<u>UML Diagram:</u>



<u>Classes and Methods:</u>

- Drunk – This class is used to define and create turtles. The user will specify how many turtles should be created. The class inherits functionality from the Turtle class and is thus a subclass of the Turtle class.
    - *Init*(*self*): – This method takes no parameters and initializes the turtle: Giving it a random position on our grid, setting its pen down and making a black dot, making its trail red, giving it the shape of a turtle, increasing the pen size, saving its starting position, and initializing its destination as its current coordinates.
    - *setAngle*(*self*): - This method takes no parameters, changes the turtle's direction, and returns the value that is used to change the direction of the turtle. The function selects a random index from 0-7 which corresponds to number representing a direction in a list called *angle*. The function then sets the heading of the turtle to the randomly selected angle and returns the angle as *angle2* for other functions to use.
    - *move*(*self*): This function takes no parameters and returns no value. It is called on an object to make it move. When called, the function does two things. First, it moves the object forward (heading has been set by *setAngle(self)*) 40 spaces. Second, it adds one to the instance variable *moves.*
    - *setX*(*self*): - Takes no parameters and sets *homeX* as the starting position x coordinate. Called only once, as soon as turtle is initialized, starting position will not change. Returns *homeX* to be used in calculating distance.
    - *SetY(self):* - Takes no parameters and sets *homeY* as the starting position y coordinate. Called only once, as soon as turtle is initialized, starting position will not change. Returns *homeY* to be used in calculating distance.
    - *destX(self):* - Takes no parameters and returns nothing, but sets destination as current position x coordinate. This function is called every time that the turtle is moved to keep up with where the turtle moves and ends its journey.
    - *destY(self):* - Takes no parameters and returns nothing, but sets destination as current position y coordinate. This function is called every time that the turtle is moved to keep up with where the turtle moves and ends its journey.
    - *returnHomeX(self):* - Takes no parameters and returns turtle's starting x coordinate.
    - *returnHomeY(self):* - Takes no parameters and returns turtle's starting y coordinate.

- *returnDestX(self): -* Takes no parameters and returns turtle's ending x coordinate.
- *returnDestY(self): -* Takes no parameters and returns turtle's ending y coordinate.
- ReturnMove(*self*): - Takes no parameters and returns the turtles number of moves
- *calcDistance(self): -* Takes no parameters and uses a formula to calculate distance. The formula calls select methods from above to get the turtle's home coordinates, set when the turtle is initiated, and the turtle's destination coordinates, updated every time the turtle moves. Returns distance.
- *DrawDistance(self): -* This function takes no parameters and simply draws the distance between the turtle's home coordinates and destination coordinates in blue. Returns nothing.

The 4 following methods are taken from the bouncing balls example from our book and are used to stop the drunk objects from going outside the grid. They all take one parameter, either the screen width or height. They use this information to calculate if the turtle is on the edge of the screen and returns TRUE if it is, FALSE if anywhere else. This boolean value is used later to decide if the turtle's heading should be changed.

- *atLeftEdge(self, screen_width):*
- *atRightEdge(self, screen_width):*
- *atTopEdge(self, screen_height):*
- *atBottomEdge(self, screen_height):*

Functions:

- *setSeconds(): -* This function takes no parameters and is used to receive input from the user. It asks the user how many seconds he/she would like each turtle to move. Precautions taken so that only a positive integer will be accepted. Returns *seconds,* which is used in a **for loop** to make the objects move a certain amount of time.
- *setKey(): -* This function takes no parameters and is used to receive input from the user. It asks the user how many creatures (turtles) he/she would like to create. Returns *key,* which is used in a for loop to initialize a certain number of turtles. Key can not be greater than 15.
- *calc_total_dist(creatures): -* Takes one parameter, *creatures,* which is a list of all the turtles. It then uses a **for loop** and the *calcDistance(self):* method to add together all the distances that the turtles traveled. Returns *total_dist.*

- *calc_avg_dist(creatures)*: - Takes one parameter, *creatures.* It calculates the total distance that the turtles traveled and then divides by the total number of turtles, effectively calculating the average distance traveled. Returns *avg_dist*.
- *calc_total_moves(creatures):* - Takes one parameter, *creatures.* It calculates the total number of moves that the turtles move

2.2. Restrictions / Limitations

There are two obvious limitations that we anticipate due to the nature of the program and the size of the screen, which is set up to 800x800.

The maximum time limit that the program can be running is set up to 300 seconds. This limitation will not allow the user to enter an unreasonably large number of seconds. The screen resolution (800x800) gives another limitation on the number of turtles that can be created by the user. The maximum number that can be created is set up to 15.

We believe that these upper limits are reasonable and justifiable numbers, given the environment of the program and the client's technical brief.

3. SOFTWARE TEST PLAN

3.1. Possible Issues, Concerns and Special Considerations

Possible Issues:

- Printing the distance travelled to the prompt makes it hard to notice.
- The program is very accurate to the point that small errors add up and make the distance travelled seem like a strange number. This can be fixed with rounding.

Possible Improvements:

- Currently all creatures are the same shape and draw at the same color.
- Currently the distance travelled is displayed in the command prompt and not in the turtle window.
- Rounding the distance travelled.

Problems solved:

- Creatures are generated at random locations though tied to the grid
- Creatures can move diagonally as well as horizontally and vertically while staying on the grid
- Creatures are prevented from moving outside of the grid area and will "bounce" back into the grid
- Creatures are a class now called "Drunk" and all associating functions are now methods in the class
- This class inherits functionality from the Turtle class and uses many of its methods.
- The program now uses moves instead of time and allows for multiple creatures.
- The program keeps track of the objects starting coordinates and ending coordinates allowing for the calculation of distance travelled

- The program also draws the distance between the starting point and its final destination.

3.2. Test & Test Results

We only have 2 variables for the user input. The first value the user should enter is the number of seconds turtles will move and the second value is the number of turtles. The maximum limit for the number of seconds is 300 (5 minutes) and the number turtles are 15. The lover limit for the turtle number is 1 and for the number of seconds is 1 as well. We got 2 types of error messages. One of them is for non- integer values and the other one is for integers for out of our range.

<u>The inputs we tried for the number of seconds giving an error message:</u>

- 'ten' (character) *
- '10 + 5' (a computation) *
- ' ' (empty) *
- '0.14' (float number) *
- '20 seconds' (character) *
- '-15' (negative number) **
- '4000' (a number higher than 300 seconds) **
- '$#@' (special characters) *
- '0'(numbers) **
- '10' (working value)

<u>Types of error messages we got for the input of number of seconds:</u>

* (Ups! I bet, it is not even a number! Let's try again!)

** (Ups! I bet, the number you have entered is out of the acceptable range! Let's try again!)

<u>The inputs we tried for the number of turtles giving an error message:</u>

- 'one'(character) *
- '#13'(character) *
- '3.14'(float) *
- '-1' (negative number) **
- '0'(number) **
- ' ' (empty) **
- '#$%' (special characters) *
- '5000' (a number higher than 15) **

- 8+9 (a computation) *
- ‘2’ (working value)

Types of error messages we got for the input of number of turtles:

* (Ups! I bet, it is not even a number! Let's try again!)

** (Ups! I bet, the number you have entered is out of the acceptable range! Let's try again!)

When we enter the correct inputs the turtle screen pops-up and we are getting the designed output.

Appendix 1 (The Program Code)

# Beginning of the code>
# >>>>>>>>>>>>>>>>>>>>

```
# -- This is an iteration of code done for CSC 500 at UNCW for the final project -- Drunken Turtles


# -- importing relevant libraries

import random

import turtle

import math

import time


# -- This is creating a new class called Drunk. It is inheriting functionality from the Turtle class and thus is a

# -- subclass of the Turtle class. It has a randomly initiated position but done so in ranges of 40 so that it always

# -- starts the center of a grid. It also has initiated y and x coordinate values and has destination y and x values

# -- that can be used to calculate the distance traveled.

class Drunk(turtle.Turtle):

    def __init__(self):

        super().__init__()

        self.speed = 0

        self.penup()

        self.setposition(random.randrange(-400, 400, 40), random.randrange(-400, 400, 40))

        self.pendown()

        self.dot(5, 'black')

        self.shape('turtle')

        self.pensize(3)

        self.pencolor('red')
        self.homeY = self.setY()

        self.homeX = self.setX()

        self.destinationX = self.destX()

        self.destinationY = self.destY()

        self.moves = 0


# -- This method chooses a random angle in 45 degree increments and sets the heading of the object to that angle as well

# -- as returning that value so that it can be used by other functions

    def setAngle(self):

        angle = [0, 45, 90, 135, 180, 225, 270, 315]

        angle2 = angle[random.randint(0, 7)]

        self.setheading(angle2)

        return angle2


# -- This function is what moves the object. The object moves in increments of 40. It also keeps track of the number of moves the object has made.

    def move(self):

            self.forward(25)

            self.moves += 1



# -- the following methods allow the programmer to set new values to the home and destination coordinates as well as

# -- return those values if need be.

    def setX(self):

        homeX = self.xcor()

        return homeX
```

```
def setY(self):
    homeY = self.ycor()
    return homeY

def destX(self):
    destX = self.xcor()
    return destX

def destY(self):
    destY = self.ycor()
    return destY

def setDestX(self):
    self.destinationX = self.destX()

def setDestY(self):
    self.destinationY = self.destY()

def returnHomeX(self):
    return self.homeX

def returnHomeY(self):
    return self.homeY

def returnDestX(self):
    return self.destinationX

def returnDestY(self):
        return self.destinationY


    def returnMove(self):
        return self.moves



# -- This method is used to calculate the distance between the two points of the object -- Its home coordinates
which
# -- are set when the object is initiated and its destination coordinates which are updated every time it moves.
    def calcDistance(self):
        distance = math.sqrt(((self.destinationX - self.homeX)**2) + ((self.destinationY - self.homeY)**2))
        return distance


# -- This function simply draws the distance between its home and destination coordinates in blue
    def DrawDistance(self):
        self.penup()
        self.goto(self.homeX, self.homeY)
        self.pencolor("blue")
        self.pendown()
        self.goto(self.destinationX, self.destinationY)
        self.penup()


# -- These functions are taken from the bouncing balls example in our book and stop the Drunk objects from going
outside
# -- of the grid. There is one small change of an additional + or -1. This is to fix a bug caused by the computer being
# -- very accurate and not rounding. It would take something like 399.9999 as not being on 400 and would allow
the Drunk
# -- object to leave the grid
    def atLeftEdge(self, screen_width):
        if (self.xcor() - 40) <= ((-screen_width / 2) + 1):
            return True
        else:
            return False

    def atRightEdge(self, screen_width):
        if (self.xcor() + 40) >= ((screen_width / 2) - 1):
            return True
        else:
            return False

    def atTopEdge(self, screen_height):
        if (self.ycor() + 40) >= ((screen_height / 2) - 1):
            return True
        else:
            return False

    def atBottomEdge(self, screen_height):
        if (self.ycor() - 40) <= ((-screen_height / 2) + 1):
            return True
        else:
            return False

# -- This function sets the amount of moves the player would like and does so in an error catching way
def setSeconds():
    flag = True
    print('Hello User! You can choose how long the program runs in a number of seconds,')
    print('but be aware that we have limited the time up to 5 minutes or 300 seconds.')
    print()
    while flag:
        try:
            seconds = int(input("Please, enter a number of seconds for program to run: "))
            if seconds > 0 and seconds<(60*5):
                flag = False
            else:
                print("Ups! I bet, the number you have entered is out of the acceptable range! Let's try again!")
                print()
        except:
            print("Ups! I bet, it is not even a number! Let's try again!")
            print()
    return seconds


# -- This function sets the "key" or how many creatures the user wants and does so in an error catching way.
def setKey():
    flag = True
    print('Well done! Now you can choose how much drunken turtles will be running around!')
    print('There is a reasonable limit of 15 turtles that can fit to our little swimming pool.')
    print()
    while flag:
        try:
            key = int(input("Please, enter a number of turtles (Max = 15): "))
            if key > 0 and key < 16:
                flag = False
            else:
                print("Ups! I bet, the number you have entered is out of the acceptable range! Let's try again!")
                print()
        except:
            print("Ups! I bet, it is not even a number! Let's try again!")
            print()
    return key


# -- This function adds together all the distances that the turtles traveled.
def calc_total_dist(creatures):
    total_dist = 0
    for x in creatures:
        total_dist += (Drunk.calcDistance(x))
    return total_dist


# -- This function calculates the average distance traveled of all the turtles.
def calc_avg_dist(creatures):
    total_dist = 0
    for x in creatures:
        total_dist += (Drunk.calcDistance(x))
    avg_dist = total_dist/len(creatures)
    return avg_dist


def calc_total_moves(creatures):
    total_moves = 0
    for x in creatures:
        total_moves += (Drunk.returnMove(x))
    return total_moves


# -- Main --
```

```
print()

print('-'*60)

print("This program simulates a random (Drunkard's) Walk")

print('-'*60)

print()


# -- These functions capture the amount of seconds and amount of creatures the user wants

seconds = int(setSeconds())

print()

key = int(setKey())

print()

print("Great! Let's Rock'n'Roll it!")

print()


# -- This sets up the windows an even 800 by 800

turtle.setup(800, 800)

screen_width = 800

screen_height = 800

window = turtle.Screen()

turtle.hideturtle()

turtle.bgpic("water.gif")

window.title('Drunk Turtles')

drunkTurtle = turtle.getturtle()


# -- This creates the grid in the same window we previously created

#grid(drunkTurtle, window)


# -- This creates an empty list -- creatures. It then iterates using a for loop in range from 0 to the "key" or
# -- the amount of creatures the player wants to be created. Each loop it iterates a new instance of the Object
"Drunk".

# -- The objects are initialized at random on the board.

creatures = []

for k in range(0, key):

    creatures.append(Drunk())


# -- This creates the movement of the creatures. For range of 0 to the amount of seconds the user wants to goes
through

# -- each Drunk object in the creature list one at a time. It generates a random angle and then checks the creature
to

# -- see if the creature is on an edge. If it is on an edge it changes the angle to the opposite direction of the edge

# -- and then moves. If it is not on an edge it moves based on the random angle generated initially. It also sets the

# -- destination value in each object to the new location it has moved too.


# -- set start time

start_time = time.time()


# -- variable used to stop movement when number of seconds reached

terminate = False


while not terminate:

        for b in creatures:

            i = Drunk.setAngle(b)

            if Drunk.atLeftEdge(b, screen_width):

                b.setheading(0)

                Drunk.move(b)

                Drunk.setDestX(b)

                Drunk.setDestY(b)
            elif Drunk.atRightEdge(b, screen_width):
                b.setheading(180)
                Drunk.move(b)
                Drunk.setDestX(b)
                Drunk.setDestY(b)
            elif Drunk.atTopEdge(b, screen_height):
                b.setheading(270)
                Drunk.move(b)
                Drunk.setDestX(b)
                Drunk.setDestY(b)
            elif Drunk.atBottomEdge(b, screen_height):
                b.setheading(90)
                Drunk.move(b)
                Drunk.setDestX(b)
                Drunk.setDestY(b)
            else:
                Drunk.move(b)
                Drunk.setDestX(b)
                Drunk.setDestY(b)
            if time.time() - start_time > seconds:
                terminate = True


# -- This function then iterates through each Drunk object in the list creatures and uses the DrawDistance function
# -- to draw a blue line from its starting point to its final destination.
for c in creatures:
    Drunk.DrawDistance(c)


# -- This puts all data for each creature in list to make it easier to print in table form
```

```
e = 1

creature_data = [['Turtle','Start x','Start y','End x','End y','Distance Traveled','Moves Made']]

for d in creatures:

   lst = ['Turtle_'+ str(e),(Drunk.returnHomeX(d)),(Drunk.returnHomeY(d)),(Drunk.returnDestX(d)),
(Drunk.returnDestY(d)),(Drunk.calcDistance(d)),(Drunk.returnMove(d))]

   creature_data.append(lst)

   e += 1


# -- This prints out the starting x and y coordinates and the ending x and y coordinates as well as the distance

# -- travelled by the creature at the end, total distance traveled by all turtles, and average distance traveled

print("\n\n")

print("Data for program run for " + str(seconds) + " seconds and with " + str(key) + " creature(s):")

dash = '-' * 80


for i in range(len(creature_data)):

   if i == 0:

     print(" ")

     print(dash)

     print('{:<4s}{:>14s}{:>9s}{:>9s}{:>9s}{:>20s}{:>12s}'.format(creature_data[i][0],creature_data[i]
[1],creature_data[i][2],creature_data[i][3],creature_data[i][4],creature_data[i][5],creature_data[i][6]))

     print(dash)

   else:

     print('{:<6s}{:>10.1f}{:>10.1f}{:>10.1f}{:>10.1f}{:>11.1f}{:>14.1f}'.format(creature_data[i][0],creature_data[i]
[1],creature_data[i][2],creature_data[i][3],creature_data[i][4],creature_data[i][5],creature_data[i][6]))


print('\nTotal Distance Traveled: ' + str('%.2f' % calc_total_dist(creatures)))

print('Average Distance Traveled: ' + str('%.2f' % calc_avg_dist(creatures)))

print('Total Moves Made: ' + str(calc_total_moves(creatures)))
```

```
# -- This makes it so the turtle window only closes after a click and will stay open for observation until that has

# -- happened.

window.exitonclick()


# >>>>>>>>>>>>>>>>>>>>
# Ending of the code>>>
```

Appendix 2 (Minutes of the Meetings)

October 25th, 2019

- - Creating the general timeline of the project.
- - Setting the priorities of tasks and subtasks.
- - Delegating tasks to group members.
- - Discussing possible issues, the group can encounter.
- - Finding a platform where everybody can meet and share their work.
- - Scheduling the next meeting.

Special Issues:

- Finding proper risks for the risk chart would be a problem because definition of risk is not defined properly.
- Initial code has been done already but classes needed to be in the code as well and the code has several problems about the movement of the cockroach.

Open Questions (Questions that are to be answered):

- What risks should have written on the risk chart?
- What kinds of risks are considered as a "project related" risks?
- Should the movement of the cockroach be in steps or minutes?

Additional Comments (Detailed Log of the Meeting):

This was the first meeting of the workgroup. The group consists of 7 people. Group members selected the main topic of the conversation as "Overall assessment of the feasibility of the task given timeline." First, group members decided to elect a team leader. That way workload can be distributed to everybody more easily and the team leader can create a timeline of the project. So, group members can finish every task at a time and focus on other problems more clearly. Group members selected Eli as the group leader. Eli started to create a timeline about our project, and we divided the task to 6 weeks.

The first week is going to be about the description part of the project. In first-week group members need to do Use case diagram, Gantt chart, and risk chart. Also selected group members for the description part will need to write a brief description of what our project is about and what our code aims to do.

The second week is going to be the coding week. In this week selected group members are going to try to finish the initial code of drunken cockroach. This week group members decided to deal with writing the code to idle and build our main part of the project. We estimated the coding part of the project would take three weeks to do which means group members should be able to finish the coding part of the project until week 5.

The fifth week is the group's debugging week. This week we decided to solve our issues about the coding part. At the end of this week, the group should run the code without any issues and errors.

Also, if there is something that needs to be added for the introduction part, selected group members will do that this week as other members finishing the code of the project.

As the group members created the timeline, members estimated time for the end of the project which is December 8 if everything goes according to the plan. Group members decided to leave a 4-day room for unexpected situations such as delayed meetings or stuck in a part of coding.

After finishing the timeline group members decided who is doing what in the project for the first part. Group members decided that Naz and Asli are going to do the description part of the project which is the first week of the project. Also, if they need help, other members agreed that everybody can help with any part of the project. The coding part of the project which is starting from week 2 and ends in week 4 is going to be completed by Evan, Ward, and Eli. Ahmet is going to do the Gantt chart and the documentation of the timeline. Cagatay is going to write all the log files, keeping track of what we should do and giving the announcements of incoming events of the group such as a meeting or a deadline. Again, these tasks are just for the first part of the project and it can be changed.

Group members started to talk about major problems that the group can encounter in the project that can affect the group's timeline drastically. Group members created a list of possible problems and decided to add them to the risk chart. One of the major problems was the concept of the risk chart such as what are the risks and what could be considered a risk. This is because members of the group were not sure about what are considered restrictions and limitations. There are not too many choices of risks for a project like this. Group members decided to talk about the coding related issues later.

Group members decided to run cockroach not with seconds but with several steps. The code will consist of several steps of directions where a cockroach is going to head. Also, instead of 4 angles members will have to move the cockroach in 8 angles. After the last lesson group members decided to add classes to the project as well. These were the only code related issues group members talked about because members had an example code of how cockroach's movement is going to be like.

For bringing the parts of the project as one document group members decided to use the school's one drive as the group's main center for documentation. That way group members don't have to meet whenever they upload their parts of the project and group can save time. Also, for the feedback and communication members opened up a slack page. The slack page will help us communicate more often and members will get constant feedback from other members of the group.

As the last thing, group members decided they will meet every 2 weeks if it is possible for every group member. The meeting lasted 1 hour.

November 8th, 2019

Agenda of Meeting:

- Bringing the code together.
- Finding errors in the code.
- Commenting about the code.
- Bringing the documentation together for the first part of the project.
- Brainstorming about the design.
- New roles assigned for the next part of the project.
- Checking on timeline to see if everything looks okay.

Special Issues:

- Brainstorming was challenging because everybody had different ideas about the code.
- Code is nearly finished but some issues needed to be solved.
- Bringing in the documentation part of the project.
- Additional design problems in the code.
- Communication of group members.

Open Questions (Questions that are to be answered):

- Should there be a background like water or lake for the turtles?
- How do we prevent turtles going off the screen?
- How do we make screen size bigger or limit the borders of screen?
- Are the variables used to make the grid dependent on the window size?
- Are there any errors we don't see in the code and is it easy to use for other users?

Additional Comments (Detailed Log of the Meeting):

This week there was no physical meeting. Group members met on Slack and talked about their work. The code of the project was almost finished, and pieces of the project are coming together. The documentation part of the project for the first step needs to be edited again. The code part of the project was shared on OneDrive. For this week's work, code was implemented in two parts. The first part consisted of brainstorming of the possible changes/improvements, revising the draft of the program and the second part consisted of applying changes such as adding more info to the output and creating a table for the output. Also, small changes were implemented to enhance the user experience. Documentation part of the project coming together as well but it is still missing small parts. Everything is going according to the timeline.

November 25th, 2019

<u>Agenda of Meeting:</u>

- Limitations of the project code
- Limiting user input (Steps and Turtle)
- Changing steps to seconds and minutes
- Changing the number of pixels
- Commenting on the feedback
- Recording number of moves / seconds
- Movement of the turtles (time and directions)
- Bringing in the final documentation
- Finding missing and additional documents required

<u>Special Issues:</u>

- Number of pixels
- Limitations of the user input (number of turtles and seconds)
- Missing documentation files
- Feedback issue on documentation
- Problems on use case diagram
- Additional documentation files required (Limitations and Restrictions and Tests File)
- Color of the turtles (randomizing the colors)

<u>Open Questions (Questions that are to be answered):</u>

- Should we limit the number of turtles user can enter?

- Should we limit the number of steps/seconds user can enter?
- What other limitations we need to add in the code in order to enhance user experience?
- Should we change the number of pixels for the better look of the output?
- Should we let user to change number of pixels?
- How we are going to do the output table that shows number of moves if we are converting the code number of moves to seconds.
- Should each turtle move the same amount of time or it the amount of time should be random?
- Can we change the color of the turtles to random?
- How the use case diagram should be? (Waiting for the feedback)

Additional Comments (Detailed Log of the Meeting):

In this meeting group members talked about mainly the restrictions and limitations of the code. The agenda was how should we limit the user input such as number of turtles and number of steps or seconds user can enter. Group members finally decided to change the steps to seconds after feedback. Another issue was to fix the pixel rate for enhanced user experience and let the user play with the pixel. After meeting additional code will be added to the project to fix pixel rate, changing steps to seconds. After changing the steps to seconds group members suggested to create a table that records number of moves and prints it to the user. Group members inquiring about randomizing the color of each turtle. After editing those into the code accordingly; the feedback will be finished for the coding phase of the project.

The documentation part of the project has not been completed yet. There are some documents missing and another two additional files required after feedback to complete the documentation (limitations and restrictions file and test file). Group members should write a document that explains every limitation and create a test log to show what problems we have encountered during the testing process. After completing the documentation file we will present to Dr.Vetter and group members will do the requested adjustments for the documentation file after feedback.