**Project group-3**

**Vamshi reddy musku , srinandh reddy gaddam , amrutha valli parangi, srinivasa rao medisetti .**

**Guide**

**Assumptions:**

Intially the project was developed using the Phython on pyscharm

For this application the feature developers are expected to be as an expert with basic phython programming that will comprise the installation of phython via pipe line and programme execution in the command line .

This program uses matplotlib for graphical

**High Level Design:**

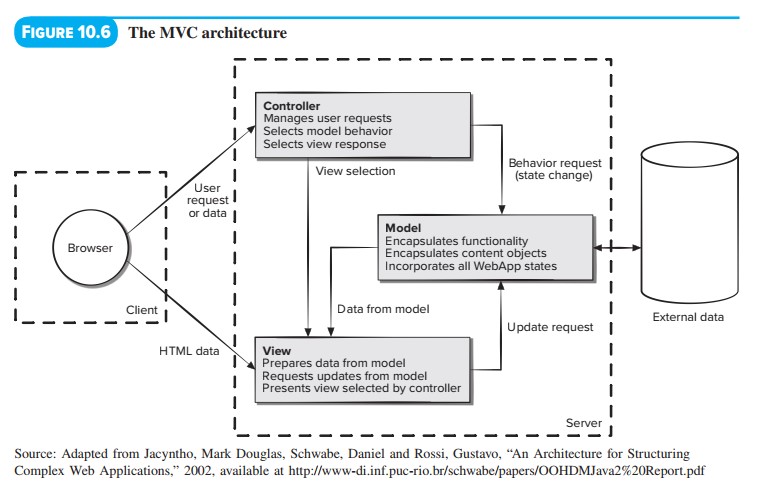


Figure 1: MVC architecture

This project will use Model-View Controller as an architectural design.

The Student users will send requests of access to the controller based on their individual grade level. The controller later manages the access requests and will selects a proper view, which is in appropriate in our case GUI will ask for the further behavior questions to run the simulation models, like the dimension of the forest grid, the number of players, and the wandering plans . Once when the controller receives those inquiries from the student users, the corresponding simulation would be triggered. Results from the new model, like as the total time to meet will be once again fed onto the view object and then reaches the students users.

For the grade students who need to study the summary statistics of the simulation. All of the results that are produced by the model, can also be kept into the internal memory, which is denoted by the ‘External data’ on the right. Since it’s a simulation experiment with independent runs, we do not need data flow from the external sources into models. If there is only one change I need to make onto the graph, I will change the double direction into a single arrow.

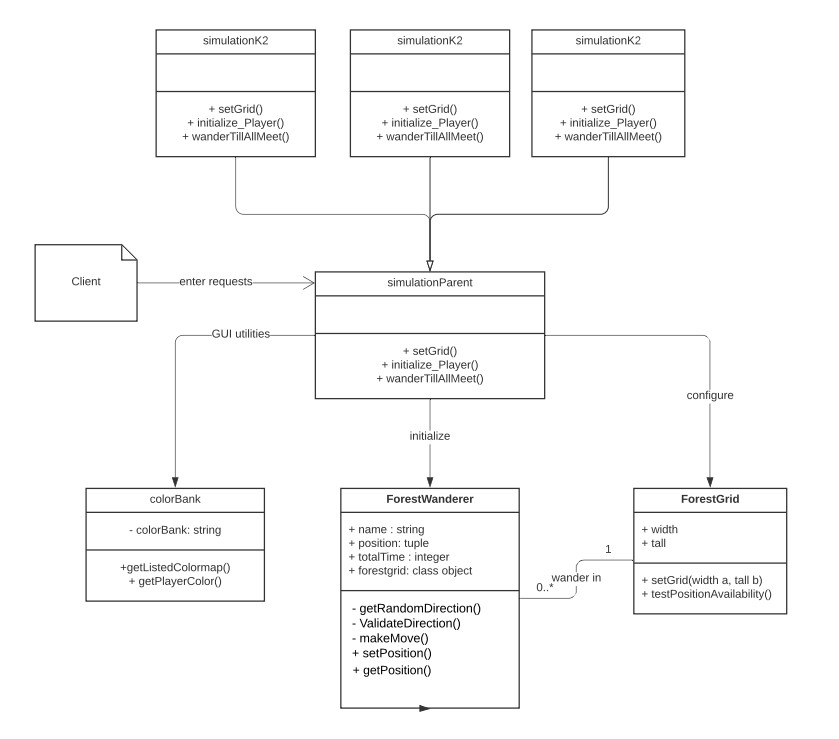
**More Indepth Designs:**

Intially the designed pattern that crossed my mind is the facade pattern. As we are working with a GUI interface that we initializes and interacts intensively with the players and with the forest Grid. I want to wrap them in one pack. As they indicated by the UML class diagram below, the simulationParent works as a parent class facade.

There is a strong association between the simulation model which it is required for K2, grade 3-5, 6-8. As a result, all the three subclasses will inherit from their parent class simulationParent. This will turns out to be a pleasant design choice, as it accommodates a very large proportion of GUI utility functions at one place , and it will also helps to reduce the presence of methods in each of the simulation subclass. However, I understand that it is essential and doesn’t help to reduce the true size of the methods, as inheritance does not equal encapsulation. This is definitely something I would like to find a better solution on when I am more experienced.

Also, my GUI relies heavily on distinct colors to denote each unique player. It would be much easier to have a static style sheet of colors, so that there would be no confusion about the color consistency. For each simulation subclass, one instance of the color scheme is enough. Thus, a singleton pattern for class colorBank will be appropriate.

Figure-2: UML Class diagram



**Installation Instructions:**

To execute the program successfully, we need to have python 3 and an additional three libraries installed. There are only 6 files required to run the applications. Place them in a location where you have the admin rights (not necessary, but recommended).

Go to the command prompt (or terminal), navigate to the folder where dependencies python files are collected.Then Launch Program command as below.

C:\Users\Software Engineering\Project>python main.py

* System Requirements:

○ Python 3 (above 3.7 is recommended) ● Library Dependencies:

○ numpy - standard libraries of Python

○ matplotlib

○ tkinter- standard libraries of Python ● Python File Dependencies:

○ main.py

○ MeetinLostForestProject.py

○ simulationParent.py

○ simulation35.py

○ simulation68.py

○ simulationK2.py

○ colorBank.py

**Appendix A. Implementation Code:**

* main.py

|  |
| --- |
| """  Project Title: Meet in Lost Forest  Project File Name: MeetinLostForest.py  Author: Vamshi; Vamshio@lewisu.edu  Course Number/Session Number: CPSC-60500-001  Submission Date : 12/14/2022  Project Explanation:  Simulation for grade k-2, 3-5, 6-8 Resources referred: None """  **import** simulationK2 **as** sk2 **import** simulation35 **as** s35 **import** simulation68 **as** s68 **import** tkinter **as** tk  **def main**():  simulationK2 = sk2.simulationK2() simulation35 = s35.simulation35() simulation68 = s68.simulation68() root = tk.Tk()  row\_idx = 0 tk.Label(root, text=" Forest Wander Simulations", font=10).grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root,  text="==================================================================#").grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text=" Please select your grade").grid(column=0, row=row\_idx)  row\_idx += 1 button = tk.Button(root, text="Grade K-2", command=**lambda**: simulationK2.init\_gui()) button.grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text="--------------------------------------").grid(column=0,  row=row\_idx) |
| row\_idx += 1 button2 = tk.Button(root, text="Grade 3-5", command=**lambda**: simulation35.init\_gui()) button2.grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text="--------------------------------------").grid(column=0,  row=row\_idx)  row\_idx += 1 button3 = tk.Button(root, text="Grade 6-8 ", command=**lambda**: simulation68.init\_gui()) button3.grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root,  text="==================================================================#").grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text="Project Description: Multiple players are wandering in the Forest  grid, trying to meet together.").grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text=" This application helps you better understand their behaviors  through graphical simulations ").grid(column=0, row=row\_idx)  row\_idx += 1 tk.Label(root, text="").grid(column=0, row=row\_idx)  row\_idx += 1 button4 = tk.Button(root, text="Exit Program", command= root.destroy, bg = '#DE3163') button4.grid(column=0, row=row\_idx) tk.mainloop()  **if** \_\_name\_\_ == "\_\_main\_\_":  main() |

* meetinLostForestProject.py

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

"""

HW1 Title: Meet in Lost Forest

HW1 File Name: MeetinLostForest.py

Author: Vamshi; Vamshilewisu.edu

Course Number/Session Number: CPSC-60500-003

Submission Date : 12/14/2022

"""

|  |
| --- |
| **import** numpy **as** np **import** random **as** random  **class ForestGrid**:  *# Constraints on width and tall, ranging from 2 - 51* MinAllowed = 2  MaxAllowed = 50  *## initialize a forest grid* **def \_\_init\_\_**(self): self.width = 0 self.tall = 0  *## takes in and set dimension of the forest grid* **def setGridDimension**(self, width, tall):  **if** (width **in** range(self.MinAllowed, self.MaxAllowed + 1) **and** tall **in** range(self.MinAllowed, self.MaxAllowed + 1)):  self.width = width self.tall = tall  **else**:  print("Error: Width and Tall should be ranging from 2 to 50.")  *## test if a proposed poistion is available to reach in the forest.*  **def testPositionAvailability**(self, positionIndices):  **if** (positionIndices[0] **in** range(0, self.tall) **and** positionIndices[1] **in** range(0, self.width)):  *# print("Attempted Position is Movable")* **return True else**:  *# print("Attempted Position is not Movable")* **return False**  *## Produce an empty forest grid, use np array* **def emptyForestMap**(self): **return** (np.zeros((self.tall, self.width), dtype=int))  **def getRandomPosition**(self):  rand\_x = random.randint(0, self.tall - 1) rand\_y = random.randint(0, self.width - 1) **return**((rand\_x, rand\_y))  **def getMultiRandomPositions**(self, num\_rand): allRandSet = set() **while**(len(allRandSet) <= num\_rand):  allRandSet.add(self.getRandomPosition())  *#print(allRandSet)* **return**(list(allRandSet))  **def getRandomDiagonalPositions**(self):  Positions\_1 = [(0,0), (self.tall - 1,self.width - 1)]  Positions\_2 = [(0,self.width - 1), (self.tall - 1,0)] |

|  |
| --- |
| **if**(random.randint(0,1)):  **return**(Positions\_1) **else**:  **return**(Positions\_2)  np.zeros((3, 2), dtype=int)[2][1]  **class ForestWanderer**:  *## const for all actions, representing the change of indices for an np array*  Stay = (0, 0)  North = (-1, 0)  South = (+1, 0)  East = (0, +1) West = (0, -1)  NorthEast = (-1, +1) NorthWest = (-1, -1) SouthEast = (+1, +1)  SouthWest = (+1, -1)  *## below two variables associate the actions with a moving index*  *## eg. 1 = North, 4 = West e.t.c*  AllPossibleMoveNames = ["Stay", "North", "South", "East", "West",  "NorthEast", "NorthWest", "SouthEast", "SouthWest"]  AllPossibleMoves = np.array([Stay, North, South, East, West,  NorthEast, NorthWest, SouthEast, SouthWest])  **def \_\_init\_\_**(self, name, forestgrid):  self.name = name self.position = () self.totalTime = 0 self.forestgrid = forestgrid self.AllActionNames = [] self.AllPositions = ()  *## Generates a random integer between 1-8 to extract actions (stay is exluded)* **def \_\_getRandomDirection**(self):  move\_idx = random.randint(1, 8)  *# print("Current position: %s" % (self.position,))*  *# print("Random Direction Picked: %s" % ( self.AllPossibleMoveNames[move\_idx] ,))* **return** (move\_idx)  **def \_\_getLastMoveIndex**(self):  **if**(self.totalTime==0):  **return**(0) **else**:  LastName = self.AllActionNames[-1] **return**(self.AllPossibleMoveNames.index(LastName))  **def \_\_getDeterminedDirection**(self): |

|  |
| --- |
| **if**(self.\_\_getLastMoveIndex() == 0):  **return**(random.randint(1, 8)) **else**:  **return**(self.\_\_getLastMoveIndex())  *## for all the moving index, proposed an new position to move* **def \_\_getAttemptPosition**(self, move\_idx):  **return** (np.add(self.position, self.AllPossibleMoves[move\_idx]))  *## \_\_ValidateDirection: validate if a moving index is feasible ## if feasible, preserve the current value, if not turn 0 (stay)* **def \_\_ValidateDirection**(self, move\_idx):  Attemp\_Position = self.\_\_getAttemptPosition(move\_idx) *## if movable* **if** (self.forestgrid.testPositionAvailability(Attemp\_Position)):  **return** (move\_idx) **else**:  *# print("Action changed to Stay")* **return** (0)  *## move to new position by changing the self.position value* **def \_\_movetoNewPosition**(self, move\_idx):  self.position = self.\_\_getAttemptPosition(move\_idx) *# print("Moved to new position: %s" % (self.position,))*  *## set position by public method, can be used to initialize starting position* **def setPosition**(self, new\_position):  self.position = new\_position self.AllPositions = np.array([self.position])  *## return position* **def getPosition**(self):  **return** (self.position)  *## reset time* **def clearTime**(self):  self.totalTime = 0  *## calcuate the total time* **def getTotalTimes**(self):  **return** (self.totalTime)  *## integral process for making an move: random an int, validate and correct the int*  *## make actual move after validation, update attributes accordingly* **def makeOneMove**(self):  *# print("Forest Wanderer: " + self.name)* move\_idx = self.\_\_getRandomDirection() corrected\_move\_idx = self.\_\_ValidateDirection(move\_idx) self.\_\_movetoNewPosition(corrected\_move\_idx) self.AllActionNames.append(self.AllPossibleMoveNames[corrected\_move\_idx]) |

|  |
| --- |
| *# print(self.AllPositions)* self.AllPositions = np.append(self.AllPositions, np.array([self.position]), axis=0) self.totalTime = self.totalTime + 1  **def makeDeterminedMove**(self):  move\_idx = self.\_\_getDeterminedDirection() corrected\_move\_idx = self.\_\_ValidateDirection(move\_idx) self.\_\_movetoNewPosition(corrected\_move\_idx)  self.AllActionNames.append(self.AllPossibleMoveNames[corrected\_move\_idx]) *# print(self.AllPositions)* self.AllPositions = np.append(self.AllPositions, np.array([self.position]), axis=0) self.totalTime = self.totalTime + 1  **def makeNoMove**(self):  corrected\_move\_idx = 0 self.\_\_movetoNewPosition(corrected\_move\_idx)  self.AllActionNames.append(self.AllPossibleMoveNames[corrected\_move\_idx]) *# print(self.AllPositions)* self.AllPositions = np.append(self.AllPositions, np.array([self.position]), axis=0) self.totalTime = self.totalTime + 1  *## print all the action name that the ForestWanderer taken* **def printPath**(self):  print("All actions for " + self.name + " are ...") print(self.AllActionNames) print(self.AllPositions)  *## frequency plot calcuates the accumulate time spent in each position of the grid* **def printFrequencyPlot**(self):  freq\_map = np.copy(self.forestgrid.emptyForestMap())  **for** time **in** range(self.totalTime + 1):  p = np.copy(self.AllPositions[time]) *# freq\_map[p[0],p[1]]* freq\_map[p[0], p[1]] = freq\_map[p[0]][p[1]] + 1  print("Travel Frequency Plot for " + self.name + " is ...") print(freq\_map)  *## active score calculates the percentage of the times ## where the ForestWanderer is making actual moves (not stay)* **def printActiveScore**(self):  activeScore = 1 - self.AllActionNames.count("Stay") / float(self.totalTime) print("Active score for " + self.name + " = {:.2%}".format(activeScore))  **def main**():  WidthA = int(input("Please input Width A (2 <= A <= 50): ")) TallB = int(input("Please input Tall B (2 <= B <= 50): ")) **while** (WidthA **not in** range(2, 51) **or** TallB **not in** range(2, 51)): |

|  |
| --- |
| print("Error:Incorrect input on Width or Tall, please check range and reenter!")  WidthA = int(input("Please input Width A (2 <= A <= 50): ")) TallB = int(input("Please input Tall B (2 <= A <= 50): "))  forestgrid = ForestGrid()  forestgrid.setGridDimension(WidthA, TallB)  Pat = ForestWanderer("Pat", forestgrid) Pat.setPosition((0, 0)) *# top right*  Chris = ForestWanderer("Chris", forestgrid)  Chris.setPosition((TallB - 1, WidthA - 1)) *# bottom left*  Final\_Situation = "B"  Final\_Meet\_Location = ()  Final\_Times = 0  Max\_TimeOut = 1000000  *## loop thru maximum allowed time* **for** times **in** range(Max\_TimeOut):  Pat.makeOneMove()  Chris.makeOneMove()  *# if Pat adn Chris meet, change the situation value,* **if** (np.array\_equal(Pat.getPosition(), Chris.getPosition())):  Final\_Situation = "A"  Final\_Meet\_Location = Pat.getPosition() Final\_Times = Pat.getTotalTimes() **break**  input("Press Enter to Show Results...") **if** (Final\_Situation == "A"):  print("####################################################") print("Situation A! Pat and Chris meet at %s" % (Final\_Meet\_Location,)) print("Total time spent is : " + str(Final\_Times)) print("####################################################")  *# Pat.printPath()*  Pat.printActiveScore() Pat.printFrequencyPlot() print("====================================================") *# Chris.printPath()*  Chris.printActiveScore() Chris.printFrequencyPlot() print("====================================================")  **else**:  print("Situation B: Pat and Chris didn't meet each other.") print("Max Time out at " + Max\_TimeOut)  *# Pat.printPath()*  Pat.printActiveScore()  Pat.printFrequencyPlot() |

|  |
| --- |
| print("====================================================")  *# Chris.printPath()*  Chris.printActiveScore() Chris.printFrequencyPlot() print("====================================================")  **def additionalExperiments**():  experiment\_id = 0  Results = np.zeros(6, dtype=int).reshape(1, 6) **for** repeat **in** range(5):  **for** length **in** range(5, 51, 5):  experiment\_id += 1  WidthA = length TallB = length  forestgrid = ForestGrid()  forestgrid.setGridDimension(WidthA, TallB)  Pat = ForestWanderer("Pat", forestgrid) Pat.setPosition((0, 0)) *# top right*  Chris = ForestWanderer("Chris", forestgrid)  Chris.setPosition((TallB - 1, WidthA - 1)) *# bottom left*  Final\_Situation = 0 *# = B = miss*  Final\_Times = 0  Max\_TimeOut = 1000000  *## loop thru maximum allowed time* **for** times **in** range(Max\_TimeOut):  Pat.makeOneMove()  Chris.makeOneMove()  *# if Pat adn Chris meet, change the situation value,* **if** (np.array\_equal(Pat.getPosition(), Chris.getPosition())):  Final\_Situation = 1 *# = A = meet* Final\_Times = Pat.getTotalTimes() **break**  result = (experiment\_id, int(repeat), WidthA, TallB, Final\_Situation,  Final\_Times)  Results = np.append(Results, np.array(result).reshape(1, 6), axis=0)  *# print(Results)* np.savetxt("Experiment.csv", Results, delimiter=",", fmt='%10.0f') print("Experiments results saved!") |
| **if** \_\_name\_\_ == "\_\_main\_\_": *## for iterative user input* main()  *## for experiment.*  run\_experiment = **False**  **if** (run\_experiment):  additionalExperiments() |

* simulationParent.py

|  |
| --- |
| **import** MeetinLostForestProject **as** fg **import** colorBank **as** cb **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** matplotlib **import** colors **import** matplotlib.animation **as** animation **from** matplotlib.backends.backend\_tkagg **import** FigureCanvasTkAgg **from** matplotlib.figure **import** Figure **import** tkinter **as** tk **import** time **from** statistics **import** mean **class simulationParent**:  colorbank = cb.colorBank() Max\_TimeOut = 1000000 meetTime = []  **def \_\_init\_\_**(self): self.forestgrid = fg.ForestGrid() self.playerlist = [] self.meetTime = []  *# set grid of the forest* **def setGrid**(self, WidthA ,TallB):  self.forestgrid = fg.ForestGrid() self.forestgrid.setGridDimension(WidthA,TallB)  *# initialize player* **def initialize\_Player**(self, number\_players):  self.playerlist = [] **for** i **in** range(number\_players):  self.playerlist.append( fg.ForestWanderer(self.colorbank.getPlayerColor(i+1), self.forestgrid) ) |

|  |
| --- |
| *# place players in random positions on the grid (no overlapping)* **def reset\_player\_position**(self):  *# randomized* initialPlayerPosition = self.forestgrid.getMultiRandomPositions(len(self.playerlist))  *## diagonal*  *# initialPlayerPosition = self.forestgrid.getRandomDiagonalPositions()* i = 0 **for** player **in** self.playerlist:  player.clearTime() player.setPosition(initialPlayerPosition[i]) i = i +1  *## all take random step to meet* **def wanderTillAllMeet**(self):  Max\_TimeOut = self.Max\_TimeOut  *## loop thru maximum allowed time* **for** times **in** range(Max\_TimeOut): TotalPosition = set() **for** player **in** self.playerlist:  player.makeOneMove() *#print(tuple(player.getPosition()))*  TotalPosition.add(tuple(player.getPosition()))  *# if meet, one position,* **if** (len(TotalPosition)==1):  self.meetTime.append(self.playerlist[0].getTotalTimes()) **break**  *# initialize GUI page* **def init\_gui**(self): self.fig, self.ax = plt.subplots(figsize=(6, 6)) self.ax.axis("off") self.root = tk.Tk() tk.mainloop()  **def exit\_button**(self): self.root.quit() self.root.destroy()  **def input\_length\_warning**(self):  self.warning\_label['text'] = "Invalid input, please re-enter"  **def input\_length\_warning\_clear**(self): self.warning\_label['text'] = "" |

|  |
| --- |
| **def place\_player\_button**(self): self.reset\_player\_position() self.plot\_static\_placement(0)  *# utility function to get plot.* **def get\_plot\_data**(self, time):  data = np.copy(self.forestgrid.emptyForestMap()) **for** i **in** range(len(self.playerlist)): player = self.playerlist[i] p = player.AllPositions[time] data[p[0], p[1]] = i + 1  plot\_data = data[::-1] **return**(plot\_data)  *# plot static plot* **def plot\_static\_placement**(self,time): plot\_data = self.get\_plot\_data(time) **if**(time == self.Max\_TimeOut):  plt.title('Failed to Meet in the maximum allowed time ' + str(self.Max\_TimeOut))  **else**:  plt.title('Wandering at Time = ' + str(time))  cmap = colors.ListedColormap(self.colorbank.getListedColormap(len(self.playerlist))) plot = self.ax.pcolormesh(plot\_data, cmap=cmap, edgecolors='k', linewidths=3) self.fig.canvas.draw() **return**(plot)  *## push for animation* **def animate**(self, i):  **while**(i < self.playerlist[0].getTotalTimes()): plot\_data = self.get\_plot\_data(i) plt.title('Wandering at Time = ' + str(i)) cmap =  colors.ListedColormap(self.colorbank.getListedColormap(len(self.playerlist))) plot = self.ax.pcolormesh(plot\_data, cmap=cmap, edgecolors='k', linewidths=3) **return** (plot)  *#self.plot\_static\_placement(i)*  **if** (i == self.playerlist[0].getTotalTimes() **and** i!=0 ):  plot\_data = self.get\_plot\_data(i) plt.title('Meet at Time = ' + str(i)) plot = self.ax.pcolormesh(plot\_data, cmap=  colors.ListedColormap(self.colorbank.getColorMeet()), edgecolors='k', linewidths=3) **return** (plot)  **if**(i == self.Max\_TimeOut):  plot\_data = self.get\_plot\_data(i) plt.title('Failed to Meet in the maximum allowed time ' + str(self.Max\_TimeOut)) **return** (self.plot\_static\_placement(i))  *# pause for animation* **def pause\_animation**(self): |
| self.ani.event\_source.stop() finalTime = self.playerlist[0].getTotalTimes()  plot\_data = self.get\_plot\_data(finalTime) plt.title('Meet at Time = ' + str(finalTime)) plot = self.ax.pcolormesh(plot\_data, cmap= colors.ListedColormap(self.colorbank.getColorMeet()), edgecolors='k', linewidths=3)  self.fig.canvas.draw() **return** (plot) |

* simulationK2.py

|  |
| --- |
| **import** simulationParent **as** parent **import** MeetinLostForestProject **as** fg **import** colorBank **as** cb **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** matplotlib **import** colors **import** matplotlib.animation **as** animation **from** matplotlib.backends.backend\_tkagg **import** FigureCanvasTkAgg **from** matplotlib.figure **import** Figure **import** tkinter **as** tk **import** time  **class simulationK2**(parent.simulationParent):  **def reset\_player\_position**(self):  *## randomized*  *# initialPlayerPosition =*  *self.forestgrid.getMultiRandomPositions(len(self.playerlist))*  *## diagonal* initialPlayerPosition = self.forestgrid.getRandomDiagonalPositions() i = 0 **for** player **in** self.playerlist:  player.clearTime() player.setPosition(initialPlayerPosition[i]) i = i +1 |

|  |
| --- |
| *# empty canvas, clear plot* **def reset\_canvas**(self): self.reset\_player\_position() self.ax.clear() self.ax.axis("off") self.fig.canvas.draw()  **def init\_gui**(self): self.fig, self.ax = plt.subplots(figsize=(6, 6)) self.ax.axis("off") self.root = tk.Tk()  tk.Label(self.root, text=" Grade K - 2 Forest Wander Simulation",font =  10).grid(column=0, row=0) tk.Label(self.root, text="Enter Square length in below box, length should be between  2- 50").grid(column=0, row=1)  self.length\_entry = tk.Entry(self.root , width = 10) self.length\_entry.grid(column=0, row=2)  input\_button = tk.Button(self.root, text="Step 1: Set square length", command =  self.input\_length\_button) input\_button.grid(column=0, row=3)  button3 = tk.Button(self.root, text="Step 2: Randomly place players (in diagonal)",  command= self.place\_player\_button) button3.grid(column=0, row=4) *#button3.place(x=0, y=30)*  button = tk.Button(self.root, text="Step 3: Push Button for simulation", command=  self.animate\_button) button.grid(column=0, row=5) *#button.place(x=0, y=0)*  self.warning\_label = tk.Label(self.root, text="",fg = 'red') self.warning\_label.grid(column=0, row=6)  canvas = FigureCanvasTkAgg(self.fig, master=self.root) canvas.get\_tk\_widget().grid(column=0, row=7)  button2 = tk.Button(self.root, text="Clear all settings", command=  self.reset\_canvas) button2.grid(column=0, row=8) *#button2.place(x=0, y=30)*  exit\_button = tk.Button(self.root, text="Exit Program", bg = "#DE3163",command=  self.exit\_button)  exit\_button.grid(column=0, row=9) |
| tk.mainloop()  *# take input by push button* **def input\_length\_button**(self):  **try**:  length = int(self.length\_entry.get()) self.setGrid(length, length)  **if**(length **in** range(2,51)):  self.input\_length\_warning\_clear() **else**: self.input\_length\_warning()  **except**:  self.input\_length\_warning()  length = int(self.length\_entry.get()) self.setGrid(length, length)  data = np.copy(self.forestgrid.emptyForestMap()) *#print(data)* plot\_data = data[::-1] self.ax.clear() plt.title('Initialize Grid') cmap = colors.ListedColormap(self.colorbank.getBackGroundColor()) self.ax.pcolormesh(plot\_data, cmap=cmap, edgecolors='k', linewidths=3)  self.initialize\_Player(2) self.reset\_player\_position() self.fig.canvas.draw()  *# control for animation* **def animate\_button**(self):  self.wanderTillAllMeet() ani = animation.FuncAnimation(self.fig, self.animate, frames = np.arange(0,  self.playerlist[0].getTotalTimes() +1, 1), interval=200, blit=**False**, repeat=**False**)  self.fig.canvas.draw() |

* simulation35.py

**import** simulationParent **as** parent **import** MeetinLostForestProject **as** fg **import** colorBank **as** cb **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** matplotlib **import** colors **import** matplotlib.animation **as** animation

|  |
| --- |
| **from** matplotlib.backends.backend\_tkagg **import** FigureCanvasTkAgg **from** matplotlib.figure **import** Figure **import** tkinter **as** tk **import** time **from** statistics **import** mean **class simulation35**(parent.simulationParent):  **def init\_gui**(self): self.fig, self.ax = plt.subplots(figsize=(6, 6)) self.ax.axis("off") self.root = tk.Tk()  row\_idx = 0 tk.Label(self.root, text=" Grade 3 - 5: Forest Wander Simulation",font = 10).grid(column=1, row=row\_idx)  row\_idx += 1 tk.Label(self.root, text="Enter width A and tall B in below boxes, both length  should be between 2- 50").grid(column=1, row=row\_idx)  row\_idx += 1 self.width\_entry = tk.Entry(self.root , width = 10) self.width\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 self.tall\_entry = tk.Entry(self.root , width = 10) self.tall\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 l3 = tk.Label(self.root, text=" Enter number of players below, the number should be  between 2 - 4") l3.grid(column=1, row= row\_idx)  row\_idx += 1 self.num\_player\_entry = tk.Entry(self.root , width = 10) self.num\_player\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 input\_button = tk.Button(self.root, text="Step 1: Set the width A and tall B,  configure number of players", command = self.input\_length\_button)  input\_button.grid(column=1, row=row\_idx)  row\_idx += 2 button3 = tk.Button(self.root, text="Step 2: Randomly place all players", command=  self.place\_player\_button) button3.grid(column=1, row=row\_idx) *#button3.place(x=0, y=30)*  row\_idx += 1 button = tk.Button(self.root, text="Step 3: Push Button for simulation", command= |

|  |
| --- |
| self.animate\_button) button.grid(column=1, row=row\_idx) *#button.place(x=0, y=0)*  row\_idx += 1 self.warning\_label = tk.Label(self.root, text='', fg = 'red') self.warning\_label.grid(column=1, row=row\_idx)  row\_idx += 1 canvas = FigureCanvasTkAgg(self.fig, master=self.root) canvas.get\_tk\_widget().grid(column=1, row = row\_idx)  row\_idx += 1 self.stat\_label1 = tk.Label(self.root, text='Total Number of Experiments: ' +  self.calculateMeetStatistics()[0] + ' | ' + \  'Average Meet Time: ' +  self.calculateMeetStatistics()[3]) self.stat\_label1.grid(column=1, row=row\_idx)  row\_idx += 1 self.stat\_label2 = tk.Label(self.root, text='Minimum Meet Time: ' +  self.calculateMeetStatistics()[1] + ' | ' + \  'Maximum Meet Time: ' +  self.calculateMeetStatistics()[2]) self.stat\_label2.grid(column=1, row=row\_idx)  row\_idx += 1 tk.Label(self.root, text='').grid(column=1, row=row\_idx)  row\_idx += 2 pause\_button = tk.Button(self.root, text="\*\*Optional: Skip simulation and show  results", bg = '#6495ED', command= self.pause\_animation)  pause\_button.grid(column=1, row=row\_idx)  *#button.place(x=0, y=0)* button2 = tk.Button(self.root, text="Clear Results", bg = '#9FE2BF',command=  self.reset\_canvas) button2.grid(column=0, row=row\_idx) *#button2.place(x=0, y=30)*  exit\_button = tk.Button(self.root, text="Exit Program", bg = "#DE3163",command=  self.exit\_button) exit\_button.grid(column=2, row=row\_idx)  tk.mainloop()  **def reset\_canvas**(self): self.meetTime = [] |

|  |
| --- |
| self.reset\_player\_position() self.ax.clear() self.ax.axis("off") self.fig.canvas.draw() self.updateStatLabels()  **def calculateMeetStatistics**(self):  **if**(len(self.meetTime) ==0 ):  **return** ['0']\*4 **else**:  stats =[len(self.meetTime), min(self.meetTime), max(self.meetTime), round(mean(self.meetTime))]  **return**([str(i) **for** i **in** stats])  **def input\_length\_button**(self):  **try**:  *# length = int(self.length\_entry.get()) # self.setGrid(length, length)*  width = int(self.width\_entry.get()) tall = int(self.tall\_entry.get()) num\_player = int(self.num\_player\_entry.get()) self.setGrid(width, tall)  **if**(width **in** range(2,51) **and** tall **in** range(2,51) **and** num\_player **in** range(2,5)):  self.input\_length\_warning\_clear() **else**: self.input\_length\_warning()  **except**:  self.input\_length\_warning()  data = np.copy(self.forestgrid.emptyForestMap()) *#print(data)* plot\_data = data[::-1] self.ax.clear() plt.title('Initialize Grid') cmap = colors.ListedColormap(self.colorbank.getBackGroundColor()) self.ax.pcolormesh(plot\_data, cmap=cmap, edgecolors='k', linewidths=3)  self.initialize\_Player(num\_player) self.reset\_player\_position() self.fig.canvas.draw()  **def updateStatLabels**(self): |
| self.stat\_label1['text'] ='Total Number of Experiments: ' +  self.calculateMeetStatistics()[0] + ' | ' + \  'Average Meet Time: ' +  self.calculateMeetStatistics()[3]  self.stat\_label2['text'] ='Minimum Meet Time: ' + self.calculateMeetStatistics()[1] + ' | ' + \  'Maximum Meet Time: ' + self.calculateMeetStatistics()[2]  **def animate\_button**(self):  self.wanderTillAllMeet() self.ani = animation.FuncAnimation(self.fig, self.animate, frames = np.arange(0,  self.playerlist[0].getTotalTimes() +1, 1), interval=100, blit=**False**, repeat=**False**)  self.fig.canvas.draw() *# print("update stat") # print(self.calculateMeetStatistics())* self.updateStatLabels() |

* simulation68.py

|  |
| --- |
| **import** simulationParent **as** parent **import** MeetinLostForestProject **as** fg **import** colorBank **as** cb **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** matplotlib **import** colors **import** matplotlib.animation **as** animation **from** matplotlib.backends.backend\_tkagg **import** FigureCanvasTkAgg **from** matplotlib.figure **import** Figure **import** tkinter **as** tk **import** time **from** statistics **import** mean  **class simulation68**(parent.simulationParent):  **def allWanderbutOne**(self):  Max\_TimeOut = self.Max\_TimeOut  *## loop thru maximum allowed time* **for** times **in** range(Max\_TimeOut): TotalPosition = set() **for** i **in** range(0,len(self.playerlist)):  **if**(i==0): |

|  |
| --- |
| self.playerlist[i].makeNoMove() *#print(self.playerlist[i].getPosition())* **else**:  self.playerlist[i].makeOneMove()  TotalPosition.add(tuple(self.playerlist[i].getPosition()))  *# if meet, one position,* **if** (len(TotalPosition)==1):  self.meetTime.append(self.playerlist[0].getTotalTimes()) **break**  **def DeterminedWanderTillMeet**(self):  Max\_TimeOut = self.Max\_TimeOut  *## loop thru maximum allowed time* **for** times **in** range(Max\_TimeOut): TotalPosition = set() **for** player **in** self.playerlist:  player.makeDeterminedMove() TotalPosition.add(tuple(player.getPosition()))  *# if meet, one position,* **if** (len(TotalPosition)==1):  self.meetTime.append(self.playerlist[0].getTotalTimes()) **break**  **def wanderPolicy**(self, policy\_number = 0):  *#print(policy\_number)* **if**(policy\_number ==1): self.allWanderbutOne()  **elif**(policy\_number == 2):  self.DeterminedWanderTillMeet() **else**:  self.wanderTillAllMeet()  **def init\_gui**(self): self.fig, self.ax = plt.subplots(figsize=(6, 6)) self.ax.axis("off") self.root = tk.Tk()  row\_idx = 0 tk.Label(self.root, text=" Grade 3 - 5: Forest Wander Simulation",font = 10).grid(column=1, row=row\_idx)  row\_idx += 1 tk.Label(self.root, text="Enter width A and tall B in below boxes, both length  should be between 2- 50").grid(column=1, row=row\_idx)  row\_idx += 1 self.width\_entry = tk.Entry(self.root , width = 10) |

|  |
| --- |
| self.width\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 self.tall\_entry = tk.Entry(self.root , width = 10) self.tall\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 l3 = tk.Label(self.root, text=" Enter number of players below, the number should be  between 2 - 4") l3.grid(column=1, row= row\_idx)  row\_idx += 1 self.num\_player\_entry = tk.Entry(self.root , width = 10) self.num\_player\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 input\_button = tk.Button(self.root, text="Step 1: Set the width A and tall B,  configure number of players", command = self.input\_length\_button)  input\_button.grid(column=1, row=row\_idx)  row\_idx += 2 button3 = tk.Button(self.root, text="Step 2: Randomly place all players", command=  self.place\_player\_button) button3.grid(column=1, row=row\_idx) *#button3.place(x=0, y=30)*  row\_idx += 1 tk.Label(self.root, text="").grid(column=1, row=row\_idx)  row\_idx += 1 l4 = tk.Label(self.root, text=" Enter Strategy Number; 1- All wander but one |  2-Determined Move (Default Random)") l4.grid(column=1, row=row\_idx)  row\_idx += 1 self.policy\_entry = tk.Entry(self.root, width=10) self.policy\_entry.grid(column=1, row=row\_idx)  row\_idx += 1 button = tk.Button(self.root, text="Step 3: Push Button for simulation", command=  self.animate\_button) button.grid(column=1, row=row\_idx) *#button.place(x=0, y=0)*  row\_idx += 1 self.warning\_label = tk.Label(self.root, text='', fg = 'red') self.warning\_label.grid(column=1, row=row\_idx)  row\_idx += 1 canvas = FigureCanvasTkAgg(self.fig, master=self.root) canvas.get\_tk\_widget().grid(column=1, row = row\_idx) |

|  |
| --- |
| row\_idx += 1 self.stat\_label1 = tk.Label(self.root, text='Total Number of Experiments: ' +  self.calculateMeetStatistics()[0] + ' | ' + \  'Average Meet Time: ' +  self.calculateMeetStatistics()[3]) self.stat\_label1.grid(column=1, row=row\_idx)  row\_idx += 1 self.stat\_label2 = tk.Label(self.root, text='Minimum Meet Time: ' +  self.calculateMeetStatistics()[1] + ' | ' + \  'Maximum Meet Time: ' +  self.calculateMeetStatistics()[2]) self.stat\_label2.grid(column=1, row=row\_idx)  row\_idx += 1 tk.Label(self.root, text='').grid(column=1, row=row\_idx)  row\_idx += 2 pause\_button = tk.Button(self.root, text="\*\*Optional: Skip simulation and show  results", bg = '#6495ED', command= self.pause\_animation)  pause\_button.grid(column=1, row=row\_idx)  *#button.place(x=0, y=0)* button2 = tk.Button(self.root, text="Clear Results", bg = '#9FE2BF',command=  self.reset\_canvas) button2.grid(column=0, row=row\_idx) *#button2.place(x=0, y=30)*  exit\_button = tk.Button(self.root, text="Exit Program", bg = "#DE3163",command=  self.exit\_button) exit\_button.grid(column=2, row=row\_idx)  tk.mainloop()  **def reset\_canvas**(self):  self.meetTime = [] self.reset\_player\_position() self.ax.clear() self.ax.axis("off") self.fig.canvas.draw() self.updateStatLabels()  *# calculate meet time statistics* **def calculateMeetStatistics**(self):  **if**(len(self.meetTime) ==0 ):  **return** ['0']\*4 **else**:  stats =[len(self.meetTime), |

|  |
| --- |
| min(self.meetTime), max(self.meetTime), round(mean(self.meetTime))]  **return**([str(i) **for** i **in** stats])  **def input\_length\_button**(self):  **try**:  *# length = int(self.length\_entry.get()) # self.setGrid(length, length)*  width = int(self.width\_entry.get()) tall = int(self.tall\_entry.get()) num\_player = int(self.num\_player\_entry.get()) self.setGrid(width, tall)  **if**(width **in** range(2,51) **and** tall **in** range(2,51) **and** num\_player **in** range(2,5)):  self.input\_length\_warning\_clear() **else**: self.input\_length\_warning()  **except**:  self.input\_length\_warning()  data = np.copy(self.forestgrid.emptyForestMap()) *#print(data)* plot\_data = data[::-1] self.ax.clear() plt.title('Initialize Grid') cmap = colors.ListedColormap(self.colorbank.getBackGroundColor()) self.ax.pcolormesh(plot\_data, cmap=cmap, edgecolors='k', linewidths=3)  self.initialize\_Player(num\_player) self.reset\_player\_position() self.fig.canvas.draw()  *# update the meet time statistics on gui* **def updateStatLabels**(self):  self.stat\_label1['text'] ='Total Number of Experiments: ' +  self.calculateMeetStatistics()[0] + ' | ' + \  'Average Meet Time: ' +  self.calculateMeetStatistics()[3]  self.stat\_label2['text'] ='Minimum Meet Time: ' + self.calculateMeetStatistics()[1] + ' | ' + \  'Maximum Meet Time: ' + self.calculateMeetStatistics()[2]  **def animate\_button**(self):  **try**: |
| policyNbr = int(self.policy\_entry.get()) self.input\_length\_warning\_clear() **except**:  self.input\_length\_warning()  self.wanderPolicy(policyNbr) self.ani = animation.FuncAnimation(self.fig, self.animate, frames = np.arange(0,  self.playerlist[0].getTotalTimes() +1, 1), interval=100, blit=**False**, repeat=**False**)  self.fig.canvas.draw() *# print("update stat") # print(self.calculateMeetStatistics())* self.updateStatLabels() |

* colorBank.py

|  |
| --- |
| **class colorBank**():  ColorBank = ['white', 'blue', 'green', 'yellow', 'purple']  **def getListedColormap**(self, num\_player):  **if**(num\_player > 4):  print("too many players") **else**:  **return**(self.ColorBank[:(num\_player+1)])  **def getPlayerColor**(self, player\_seq):  **if**(player\_seq > 3):  print("too many players") **else**:  **return**(self.ColorBank[player\_seq])  **def getColorMeet**(self):  **return**(['white','red'])  **def getBackGroundColor**(self): **return**(self.ColorBank[0]) |