# Answers and Solutions

## Complete UML Diagram

# 

## Programming Exercises (Solutions)

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 1 | def Main():  MenuOption = 0  while MenuOption != 4:  print("Predator Prey Simulation Main Menu")  print()  print("1. Run simulation with default settings")  print("2. Run simulation with custom settings")  print("3. Rabbit Paradise")  print("4. Exit")  print()  MenuOption = int(input("Select option: ")) | 5 marks   * *1 mark for changing 3 to 4 in while loop condition* * *1 mark for print statement for rabbit paradise* * *1 mark for changing 3 to a 4 in print statement for exit* * *1 mark for options 3 and 4 displayed in correct order* * *1 mark for screen capture (1)* |
| 2 | MenuOption = int(input("Select option: "))  if MenuOption == 1 or MenuOption == 2 or MenuOption == 3:  if MenuOption == 1:  LandscapeSize = 15  InitialWarrenCount = 5  InitialFoxCount = 5  Variability = 0  FixedInitialLocations = True  elif MenuOption == 3:  LandscapeSize = 20  InitialWarrenCount = 20  InitialFoxCount = 0  Variability = 1  FixedInitialLocations = False  else:  … | 5 marks   * *1 mark for changing IF statement to allow MenuOption to be 3* * *1 mark for adding an ELIF and correct condition* * *2 marks for assigning variables the correct values  (–1 per mistake)* * *1 mark for screen capture (1)* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 3 | while (self.\_\_WarrenCount > 0 or self.\_\_FoxCount > 0) and MenuOption != 5:  print()  print("0. Advance 10 time periods hiding detail")  …  …  MenuOption = int(input("Select option: "))  if MenuOption == 0:  self.\_\_TimePeriod += 10  self.\_\_ShowDetail = False  for x in range(10):  self.\_\_AdvanceTimePeriod() | 7 marks   * *6 marks for changes shown (1 mark per line)* * *1 mark for screen capture* |
| 4 | def \_\_init\_\_(self, Variability, ParentsReproductionRate = 1.2, genderRatio = 50):  self.\_\_DEFAULT\_LIFE\_SPAN = 4  ..  ..  if random.randint(0, 100) < genderRatio:  self.\_\_Gender = Genders.Male  … | 2 marks   * *2 marks for changes shown (1 mark each)* |
| 5 | class Fox(Animal):  …  …  self.\_\_FoodUnitsConsumedThisPeriod = 0  if (random.randint(0, 3) < 2):  self.\_\_Gender = Genders.Female  else:  self.\_\_Gender = Genders.Male  def ReproduceThisPeriod(self):  if self.\_\_Gender == Genders.Male:  return False  else:  REPRODUCTION\_PROBABILITY = 0.25  … | 12 marks   * *4 marks for successfully adding gender to a fox  (1 mark per line)* * *3 marks for stopping male foxes from breeding* * *4 marks for outputting the gender of a fox in an inspection* * *1 mark for screen capture* |
| 5 (cont.) | def Inspect(self):  super(Fox, self).Inspect()  print("Food needed", self.\_\_FoodUnitsNeeded, "", end = "")  print("Food eaten", self.\_\_FoodUnitsConsumedThisPeriod, "", end = "")  if self.\_\_Gender == Genders.Female:  print("Gender Female")  else:  print("Gender Male")  print() |  |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 6 | class GiantWarren(Warren):  def \_\_init\_\_(self, Variability, RabbitCount = 50):  self.\_\_MAX\_RABBITS\_IN\_WARREN = 200  super()  self.\_\_RabbitCount = RabbitCount  …  def \_\_CreateLandscapeAndAnimals(self, InitialWarrenCount, InitialFoxCount, FixedInitialLocations):  …  if FixedInitialLocations:  …  self.\_\_Landscape[10][3].Warren = Warren(self.\_\_Variability, 52)  self.\_\_Landscape[11][4].Warren = GiantWarren(self.\_\_Variability, 115)  self.\_\_WarrenCount = 6  self.\_\_Landscape[2][10].Fox = Fox(self.\_\_Variability)  ***Plus:*** *Warren instance variables need to be protected and  not private (done by changing \_\_ to \_ ):*  self.\_MAX\_RABBITS\_IN\_WARREN = 99  self.\_RabbitCount = RabbitCount  self.\_PeriodsRun = 0  self.\_AlreadySpread = False  self.\_Variability = Variability  self.\_Rabbits = [] | 11 marks   * *4 marks for GiantWarren class (1 for signature plus  3 for each other highlighted change)* * *3 marks for changes to CreateLandscapeAndAnimals (1 for creating a GiantWarren, 1 for putting it in the correct position, 1 for changing the warren count)* * *3 marks for changing Warren instance variables to protected and not private  (1 mark for each pair of variables changed)* * *1 for screen capture showing the GiantWarren* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 7 | class Den:  def \_\_init\_\_(self):  self.\_\_foxes = 0  def spawn(self):  self.\_\_foxes += 1  return Fox(50)  def getSymbol(self):  return ("D" + str(self.\_\_foxes))  class Location:  def \_\_init\_\_(self):  self.Fox = None  self.Warren = None  self.Den = None  def \_\_DrawLandscape(self):  …  for x in range (0, self.\_\_LandscapeSize):  if not self.\_\_Landscape[x][y].Warren is None:  if self.\_\_Landscape[x][y].Warren.GetRabbitCount() < 10:  print(" ", end = "")  print(self.\_\_Landscape[x][y].Warren.GetRabbitCount(), end = "")  else:  print(" ", end = "")  if not self.\_\_Landscape[x][y].Fox is None:  print("F", end = "")  elif not self.\_\_Landscape[x][y].Den is None:  print(self.\_\_Landscape[x][y].Den.getSymbol(), end = "")  else:  print(" ", end = "")  print("|", end = "")  print() | 18 marks   * *8 marks for Den class  (1 per line; if code shorter or more efficient than example, 8 can still be awarded)* * *1 mark for change to Location class* * *2 marks for changes to DrawLandscape* * *1 mark for adding a Den in CreateLandscape* * *5 marks for adding the spawn command to AdvanceTimePeriod* * 1 mark for *screen capture* |

|  |  |  |
| --- | --- | --- |
| 7 (cont.) | def \_\_CreateLandscapeAndAnimals(self, InitialWarrenCount, InitialFoxCount, FixedInitialLocations):  …  if FixedInitialLocations:  self.\_\_Landscape[1][1].Warren = Warren(self.\_\_Variability, 38)  self.\_\_Landscape[2][8].Warren = Warren(self.\_\_Variability, 80)  self.\_\_Landscape[9][7].Warren = Warren(self.\_\_Variability, 20)  self.\_\_Landscape[10][3].Warren = Warren(self.\_\_Variability, 52)  self.\_\_Landscape[13][4].Warren = Warren(self.\_\_Variability, 67)  self.\_\_Landscape[11][4].Warren = GiantWarren(self.\_\_Variability, 115)  self.\_\_WarrenCount = 6  self.\_\_Landscape[2][10].Fox = Fox(self.\_\_Variability)  self.\_\_Landscape[6][1].Fox = Fox(self.\_\_Variability)  self.\_\_Landscape[8][6].Fox = Fox(self.\_\_Variability)  self.\_\_Landscape[11][13].Fox = Fox(self.\_\_Variability)  self.\_\_Landscape[12][4].Fox = Fox(self.\_\_Variability)  self.\_\_FoxCount = 5  self.\_\_Landscape[2][3].Den = Den()  else:  …  def \_\_AdvanceTimePeriod(self):  NewFoxCount = 0  if (self.\_\_TimePeriod % 3) == 0:  x = random.randint(0, self.\_\_LandscapeSize - 1)  y = random.randint(0, self.\_\_LandscapeSize - 1)  self.\_\_Landscape[x][y].Fox = Den.spawn(self.\_\_Landscape[2][3].Den)  print("Fox spawned at "+str(x)+","+str(y))  if self.\_\_ShowDetail:  print()  for x in range (0, self.\_\_LandscapeSize): |  |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 8 | class Fox(Animal):  \_TotalDeadFoxes = 0  \_TotalFoxAge = 0    def \_\_init\_\_(self, Variability):  …  …  if (random.randint(0, 3) < 2):  self.\_\_Gender = Genders.Female  else:  self.\_\_Gender = Genders.Male  def getLifeExpect(self):  if Fox.\_TotalDeadFoxes > 0:  return float(Fox.\_TotalFoxAge/Fox.\_TotalDeadFoxes)  else:  return self.\_\_DEFAULT\_LIFE\_SPAN  def \_\_DrawLandscape(self):  …  …  if not self.\_\_Landscape[x][y].Fox is None:  print("F", end = "")  lifeExpect = self.\_\_Landscape[x][y].Fox.getLifeExpect()  else:  print(" ", end = "")  print("|", end = "")  print()  print("The average life expectancy of a fox stands at " + str(lifeExpect)) | 14 marks   * *2 marks for changes to DrawLandscape (1 for code, 1 for correct indentation and sequence)* * *2 marks for class variables in Fox* * *5 marks for creating getLifeExpect* * *2 marks for screen captures* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 9 | class Simulation:  …  …  print("5. Exit")  print("6. Find biggest warren")  print()  …  if not self.\_\_Landscape[x][y].Warren is None:  self.\_\_Landscape[x][y].Warren.Inspect()  self.\_\_ViewRabbits = input("View individual rabbits (y/n)? ")  if self.\_\_ViewRabbits == "y":  self.\_\_Landscape[x][y].Warren.ListRabbits()  if MenuOption == 6:  self.findBiggest()  input()  def findBiggest(self):  biggestX = -1  biggestY = -1  biggestSize = -1  for x in range (0, self.\_\_LandscapeSize):  for y in range (0, self.\_\_LandscapeSize):  if not self.\_\_Landscape[x][y].Warren is None:  if biggestSize < self.\_\_Landscape[x][y].Warren.GetRabbitCount():  biggestSize = self.\_\_Landscape[x][y].Warren.GetRabbitCount()  biggestX = x  biggestY = y  print("Biggest warren at (" + str(biggestX) +"," + str(biggestY) + ")") | 12 marks   * *1 mark for Print statement in menu* * *2 marks for IF statement and procedure call when a 6 is entered* * *1 mark for findBiggest signature* * *1 mark for creating and initialising variables (to a sentinel value) to store data about the current biggest warren* * *2 marks for x and y loops* * *1 mark for checking that a warren is stored at x,y* * *1 mark for checking if warren is bigger than current biggest* * *1 mark for assigning x,y, and size of a new biggest warren* * *1 mark for displaying correct message* * *1 mark for screen capture* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 10 | def CalculateNewAge(self):  self.\_Age += 1  if self.\_Age >= self.\_NaturalLifespan:  self.\_IsAlive = False  self.\_ProbabilityOfDeathOtherCauses += 0.1 | 2 marks   * *1 mark for incrementing self\_ProbabilityOfDeathOtherCause by 0.1 when rabbits age* * *1 mark for screen capture showing rabbits with different death probabilities* |
| 11 | *Changes to class Warren:*  def getRabbits(self):  return self.\_Rabbits  *Changes to class Rabbit:*  def getAge(self):  return self.\_Age  class Simulation:  …  …  print("6. Find biggest Warren")  print("7. Inspect all rabbits")  print()  …  …  if MenuOption == 6:  self.findBiggest()  if MenuOption == 7:  self.sortAndListRabbits()  input()  def sortAndListRabbits(self):  *#create a list to store all rabbits*  allRabbits = []  theRabbits = []  *#get all the rabbits from all the warrens and add to the list*  for x in range (0, self.\_\_LandscapeSize):  for y in range (0, self.\_\_LandscapeSize):  if not self.\_\_Landscape[x][y].Warren is None:  allRabbits.extend(self.\_\_Landscape[x][y].Warren.getRabbits())  *#remove "none" values*  for x in range(len(allRabbits)):  if not allRabbits[x] is None:  theRabbits.append(allRabbits[x])  *#bubble sort the rabbits list*  for passnum in range(len(theRabbits)-1,0,-1):  for i in range(passnum):  if theRabbits[i].getAge() < theRabbits[i+1].getAge():  temp = theRabbits[i]  theRabbits[i] = theRabbits[i+1]  theRabbits[i+1] = temp  *#display all the rabbits*  for x in range(len(theRabbits)):  theRabbits[x].Inspect() | 18 marks   * *2 marks for creating getRabbits in Warren (1 per correct line)* * *2 marks for creating getAge in Rabbit (1 per correct ine)* * *3 marks for changes to the menu in simulation (1 per correct line)* * *10 marks for sortAndListRabbits procedure. Minus 1 per mistake. If NOT a bubble sort then max. 5 marks can be awarded for this part of the task.* * *1 mark for screen capture* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 12 | class Simulation:  …  …  self.\_\_CreateLandscapeAndAnimals(InitialWarrenCount, InitialFoxCount, self.\_\_FixedInitialLocations)  self.\_\_Landscape = []self.\_\_WarrenGraph = WarrenGraph()  theNode = Node(1,1,2,8,9,7)  self.\_\_WarrenGraph.addNode(theNode)  theNode = Node(2,8,13,4,1,1)  self.\_\_WarrenGraph.addNode(theNode)  theNode = Node(9,7,1,1,13,4)  self.\_\_WarrenGraph.addNode(theNode)  theNode = Node(13,4,9,7,2,8)  self.\_\_WarrenGraph.addNode(theNode)  self.\_\_DrawLandscape()  …  …  print("7. Inspect all rabbits")  print("8. Display adjacency list")  print()  MenuOption = int(input("Select option: "))  if MenuOption == 8:  self.\_\_WarrenGraph.adjList()  class Node:  def \_\_init\_\_(self, selfX, selfY, leftBranchX, leftBranchY, rightBranchX, rightBranchY):  self.selfX = selfX  self.selfY = selfY  self.leftBranchX = leftBranchX  self.leftBranchY = leftBranchY  self.rightBranchX = rightBranchX  self.rightBranchY = rightBranchY | 22 marks   * *5 marks for creating and populating graph (1 for graph,  1 per node)* * *3 marks for changes to menu* * *6 marks for Node class (1 for signature, 2 for constructor,  3 marks for getCoord (1 for  each arg))* * *7 marks for WarrenGraph (1 for class signature, 2 for constructor, 2 for addNode procedure, 2 for adjList procedure)* * *1 mark for screen capture* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 12 (cont.) | def getCoord(self, arg):  X = -1  Y = -1  if arg == "s":  X = self.selfX  Y = self.selfY  if arg == "l":  X = self.leftBranchX  Y = self.leftBranchY  if arg == "r":  X = self.rightBranchX  Y = self.rightBranchY  return (X,Y)  class WarrenGraph:  def \_\_init\_\_(self):  self.\_\_nodes = []  def addNode(self, theNode):  self.\_\_nodes.append(theNode)  def adjList(self):  for Node in self.\_\_nodes:  print(Node.getCoord("s") ,": ", Node.getCoord("l") ,",", Node.getCoord("r")) |  |
| 13 | class Simulation:  …  …  print("8. Display adjacency list")  print("9. Display adjacency matrix")  print()  MenuOption = int(input("Select option: "))  if MenuOption == 9:  self.\_\_WarrenGraph.adjMatrix() | 17 marks   * *3 marks for changes to menu* * *13 marks for creating adjMatrix (1 mark for displaying column headings, 1 mark for displaying row labels correctly, 3 marks for efficient code (loops), 2 marks per correct row displayed). NO CREDIT SHOULD BE GIVEN FOR HARD-CODED SOLUTIONS – the adjacency matrix must be calculated each time by the computer.* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 13 (cont.) | *Added to class WarrenGraph:*  def adjMatrix(self):  #column heading  headingList = []  heading = "\t"  for Node in self.\_\_nodes:  headingList.append(Node.getCoord("s"))  heading += "\t" + str(Node.getCoord("s"))  print(heading)  #rows  for Node in self.\_\_nodes:  row = ""  row += (str(Node.getCoord("s")) + "\t:\t")  for item in headingList:  if (Node.getCoord("l") == item) or (Node.getCoord("r") == item):  row += " 1"  row += "\t"  print(row) | * *1 mark for screen capture* |
| 14 | def adjMatrix(self):  …  …  for item in headingList:  if (Node.getCoord("l") == item) or (Node.getCoord("r") == item):  x1, y1 = Node.getCoord("s")  if (Node.getCoord("l") == item):  x2, y2 = Node.getCoord("l")  else:  x2, y2 = Node.getCoord("r")  distance = math.sqrt((pow(x1 - x2, 2) + pow(y1 - y2, 2)))  row += " " + str(round(distance,1))  row += "\t"  print(row) | 9 marks   * *1 mark for getting the x,y coordinates of the starting point* * *4 marks for IF statement to distinguish between whether node is left or right branch and getting the cords (must be inside IF statement already there)* * *2 marks for applying Pythagoras correctly (there are several ways to do this, doesn't need to match example above; award 1 mark for a good attempt)* * *1 mark for rounding to 1dp* * *1 mark for screen capture* |

|  |  |  |
| --- | --- | --- |
| Q | Example Solution | Suggested Marks |
| 15 | class Simulation:  …  …  print("9. Display adjacency matrix")  print("10. Is there a route")  print()  MenuOption = int(input("Select option: "))  if MenuOption == 10:  self.\_\_WarrenGraph.isRoute()  if MenuOption == 9:  …  def isRoute(self):  #get coordinates of warrens  startX = int(input("Enter x coordinate of Warren 1"))  startY = int(input("Enter y coordinate of Warren 1"))  finishX = int(input("Enter x coordinate of Warren 2"))  finishY = int(input("Enter y coordinate of Warren 2"))  route = False  #find start  for Node in self.\_\_nodes:  checkX, checkY = Node.getCoord("s")  if (checkX == startX) and (checkY == startY):  checkX, checkY = Node.getCoord("l")  if (checkX == finishX) and (checkY == finishY):  route = True  checkX, checkY = Node.getCoord("r")  if (checkX == finishX) and (checkY == finishY):  route = True  if route == True:  print("There is a route between the warrens")  else:  print("There is no route between the warrens") | 13 marks   * *3 marks for changes to menu* * *8 marks for isRoute  (2 marks for getting each set of coordinates, 1 mark for loop that will check all warrens in graph, 1 mark for IF statement that find the node with correct start coordinates, 1 mark for checking left branch, 1 mark for checking right branch, 2 marks for IF statement with correct output statements)* * *2 marks for screen captures* |

# Appendi