Tianyi Yang
CS 6310
Assignment 4

- How do you track the current direction and location of each mower?
    - ➢ Since the mover don't know the absolute position of themselves, we have to track each mover's position in Lawn Class, we have variables called moverX and moverY, and their data structure types are both integer array which the mover's Id will be represented as index. The absolute position value either X or Y will be stored in this index slot in an array. In mover class, we have a public variable called curDirection which represent the current direction of that mover facing to. If we want to know the mover's current direction in Lawn class, we can access that mover through variable movers which recorded all instances of Mover class, then get access to its variable curDirection.
- How do you track the locations of gophers?
    - ➢ We can track locations of gophers through the variable xPostion and yPosition in Gopher Class. In Lawn class, we have access to get Gopher instance through variable gophers, and according to the gopher's id, we can get instance through calling gophers[gophersId]. Later we just use variable xPostition and yPosition inside Gopher class to get x and y position of this Gopher.
- How do you track the how much of the grass has been cut so far?
    - ➢ In Lawn Class, we have a function named grassHasBeenCut(). In Main class, we have instance of the Lawn class, so we could track amount of grasses that have been cut already through lawn.grassHasBeenCut() which will return the integer value of a specific amount. In grassHasBeenCut() function, we basically iterate all slots in the lawn(means iterate variable lawnInfo: int[][]), and check Code of each slot. First we have a local variable result, and set it equal to 0, and later we do the iteration, If the slot code equal to either empty_Code or gopher_Empty_Code, we let res = res + 1. At last, we return the result.
- How do you update the simulation state for each of the varied actions?
    - ➢ There are variables called curDirections, lastScan, and lastAction in each mover object. For each move() function, there is a switch(direction) in it. So each time we use move() function, we have to get class variable curDirection. According to different String of directions, we can manipulate x and y which are represents current positon in lawnInfo. After call either cscan() or lscan() we will record the result to trackScanResult in Lawn class and update variable lastScan in Mover class. After calling steer(), we will update the variable curDirection in Mover class, and change value in trackNewDirection in Lawn Class.
- How do you determine the appropriate output for a cscan() action?
    - ➢ We check the 8 surrounding place around the specific mower, since in Lawn Class, we have access to get mover's absolute position through

variable moverX and moverY, so we can calculate the 8 surrounding slot positions. Inside the variable lawnInfo, it stored current condition of each slot. If the slot is Empty, the slot will contain value empty_code. If the space is empty and have a gopher on it, the slot will contain value gopher_Empty_Code. If the space is not empty and have a gopher on it, the slot will contain value gopher_Grass_Code. And so on. That's mean each space condition will be recorded in Lawn Classes.

- How do you determine the appropriate output for an lscan() action?
  - ➢ We check all linear slots which specific mower facing to. Since in Lawn Class, we can get mover's absolute position through variable moverX and moverY. We also can get mover's current orientation through getting variable movers and find that specific mover later find it's own variable curDirection. Like the question above, Inside the variable lawnInfo, it stored current condition of each slot. If the slot is Empty, the slot will contain value empty_code. If the space is empty and have a gopher on it, the slot will contain value gopher_Empty_Code. If the space is not empty and have a gopher on it, the slot will contain value gopher_Grass_Code. And so on. So we keep asking for the slot condition until it's show us that out of space of this lawnInfo.
- How do you determine when the simulation should be halted?
  - ➢ We have an instant of lawn object in Main class. Inside Main class and inside main function, we use lawn variable to access the maxTurn through asking Lawn class variable turnLimit. Later we do a for loop, from 0 to turn limit, and for each turn(inside the for loop) we keep ask the function isAllGood(). If the function return true, that's mean it's a valid turn, we can continue the next turn until the for loop ending , we halted the simulation. Otherwise, it's mean the simulation halted since some other reasons, and the specific reason that cause simulation halted we can find it in isAllGood() function. For example, Inside the isAllGood() function, we keep asking some functions which appears inside Lawn class, moverAllStopped(), grassHasBeenCut(). If all mover stopped(check each mover's moveStatue, if 0 valid, otherwise it could be out of energy, broken, or crashed) we halt the simulation, Also if the the amount that grassBeenCut() returned equals to the amount initialGrass() returned that's mean all grasses have been cut. Again we halted the simulation.
- How do you keep track of the knowledge needed to display the final report?
  - ➢ In Main class, inside main function, after the simulation is halted, it will call finalReport(completeTurn : int) function in Lawn class while it will pass the amount turn we already pass so far as parameter. As we mentions before, we have a for loop inside the main function, from 0 to limit turns, So we could know the how many turns we gone so far. Inside the

finalReport(completeTurn : int) function, we calculate the lawn size through variable lawnWeight, and lawnHeight. We also could get initial grass and grass have been cur through initialGrass() and grassHasBeenCut(). So we could print the final result with those information.

• How do you keep track of the partial knowledge collected by each mower?

- ➢ For each mover object, we have a variable called localMap and the data structure for this is Map<String, Integer>. This variable is really important since it record all the path it passed through before. Since the mover don't know the absolute position for it's self, so the position we stored as key in map is the relative position to the mover initial position. And the value we stored is the current condition of this slot.
- ➢ For example, wherever the mover start with, we see that mover start at [0][0]. And we put {"0:0", charging_Pad_Code) into variable localMap. And after either call lscan() or cscan(), we will record information around or linear into localMap since according to the mover's current orientation, we can calculate the relative position of around or linear slots. This is easy for future mover's communication and for lower energy (using BFS strategy to find closed way to charging pad without using energy).So the key point for this data structure is we only store the relative position to mover itself, since the mover do not know how big is the stage.

• How do you manage collaboration between the mowers?

- ➢ We use mover.talkTo(Mover, Map<String, Integer>, int[]> function to implement communication between multiple movers. Mover will start to talk/collaboration when it find there is one or more mover around after cscan(). That's mean the another mover is one step far from that mover. Therefore after move's each cscan() action, we make a judgment to see if there is any mover around, if Yes, we iterator mover's x and y position in Lawn Class, to see which specific mover currently occupy that slot. And later, we use mover.talkTo(Mover, Map<String, Integer>, int[]). We have to use this function twice for each mover found, since we are letting both mover update their info. For example (
moverA.talkTo(moverB, moverB.localMap, int[]
moverB.talkTo(moverA, moverA.localMap, int[])
- ➢ The Key for this part is how to update variable A.localMap to access more information according to B.localMap. This brings us back to math problem. Let's say, since the position stored in those variable is relative to their personal mover, So we can not just copy entry from A map to B map. However, since moverA and moverB are close by one step, we can easily calculate the relative positon of moverB to moverA . (Then we calculate

the difference between relative positon of B to A and relative position of B to B). Later, for each piece of entry in map, we can add this difference, and stored the answer to own map. That's implement the communication of two movers.

• How do you determine the next action for a mower?

> ➢ We have a couple of options action for mower, in Random Strategy, the action is picked by random. In non-random Strategy, mover will record past action they did in variable lastScan. move() always called after steer(), and steer() called after we found we have grass around through variable localMap or after cscan() called. Before move action, we usually check action cscan(), to check if there is non-gopher-grass slot around. However, before cscan() function called, we usually check localMap, to make sure we have valid grass slot to go without called cscan() to save energy. Also, if we check the mover energy is lower than 30%, we will call a function bfsFindPath(), and it's return List<int[]> which is the way to closed charging pad. And we set variable onTheWayToCharging to True. Before mover did action, we will check this variable, if this variable is equal to true, we will only do the action move(), and the way is stored in variable pathToCharing;