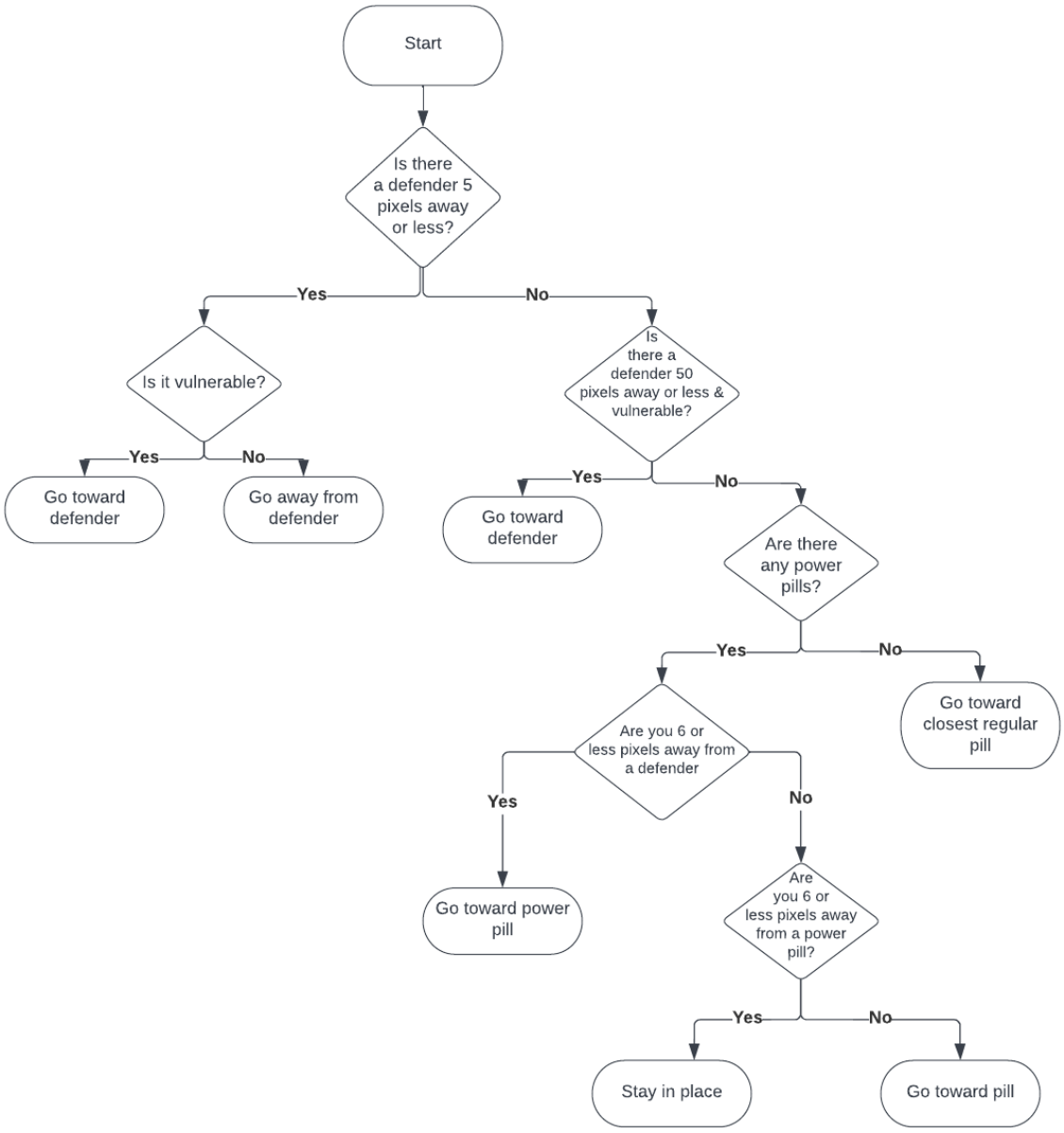
Design & Post-Mortem Document

a) First, the attacker will prioritize defenders nearby. Using the location of the attacker and defender (getLocation()), and the distance between them (getPathDistance), we see if the defender is 5 or less pixels away. If so, the isVulnerable() method is used to determine if the defender can be eaten. If the defender is vulnerable, the attacker goes toward it If not, it goes the opposite direction. These directions are defined using the getNextDirection and specifying true for toward the defender and false for away. If there is not a defender close by, we check if there is a vulnerable defender that the attacker could chase after. If there is a defender that is 50 pixels or less and is vulnerable (also found using getPathDistance & isVulnerable), then the attacker goes toward that defender. If none of these possibilities are true, the attacker focuses on its second priority: power pills. We use getPowerPillsList() and make sure the size of it is >0. This ensures there are power pills on the screen to go toward. getTargetNode and getTargetActor are used to return the nodes for the closest defender and power pill. If the defender is 6 or less pixels away (getPathDistance) from the attacker, the attacker goes toward the power pill (getNextDirection). If not, we look at how close the attacker is to the pill. If the getPathDistance between a power pill and the attacker returns 6 or less, it continuously switches directions, which results in the hovering effect of the attacker. If it's more than 6 pixels away, it will continue to travel toward the power pill. Finally, once all of the power pills have been eaten by the attacker, its third priority is to finish eating the other pills. Running away from nearby defenders has precedence over eating the pills. We use getPillList() and getTargetNode to find the closest pill, then getNextDirection sends the attacker toward that pill.

See diagram on next page.



b) The most abundant failure that I ran into during this project was calling functions with the wrong data types or interfaces. This happened quite often and most of my time on this project was spent debugging this issue and trying to convert and call the right data types and interfaces to receive the information I wanted. My first strategy was to eat all the power pills, and then the regular pills. This implementation was a success, but my attacker did not go after the vulnerable defenders once he ate a pill. I tried to find a way for the attacker to eat the defenders, but I couldn't call the location of the closest defender the same way that I could call the location of the closest power pill. This led to me having to loop through the list of defenders and find out which ones were vulnerable and/or nearby. Once I got this to work, I used TA Marco's strategy of waiting by the power pill and then eating it so that I can immediately get the vulnerable defender. This strategy improved my score a lot, but looping through the defenders didn't make

sense within the statements I had created for eating power pills. I decided to prioritize eating vulnerable defenders/running away from non-vulnerable ones, so I put the loop at the beginning of the code. Once I did this, the attacker priorities went as follows: 1) if there are defenders nearby, eat the vulnerable ones & run away from the non-vulnerable ones 2) go to a power pill and wait for a defender to come 3) once the power pills were gone, eat the rest of the pills. This code was a success in implementation, but I was consistently scoring an average of approximately 6084. In order to see what was going wrong, I watched the game progress and saw there were times when my attacker was stuck at a wall and not moving. After a lot of debugging and trial and error, I realized the issue was coming from an if statement in which I was checking if a defender was <= 5 pixels away. When the defenders were in the lair, they were returning a -1, which was technically <= 5, so the defender wasn't moving and therefore the attacker couldn't go toward or away from it. I simply added another condition to the if statement that the distance had to also be >= 0, and this fixed that issue. I scored over 6800 points and completed the task.

c) This project was very frustrating at first due to my lack of understanding of the different data types and how they all work together. It took a lot of logical processes and I found myself asking other people what their strategies are when playing PacMan. I took their personal strategies when playing the game and turned it into a set of decisions for the computer to follow. This project took me multiple days to complete, and I found that when I got frustrated, stepping away for a brief period and coming back with a clear head gave me a better approach to the problem. The tutorial videos provided by the TAs and the review during discussion section were immensely helpful and I don't think I would have been as successful on this project without them. I learned how difficult it was to work with code that I didn't write myself, because it takes just as much, if not more, time to understand the code given and how it all works together so that you can manipulate and implement it in your own work. I realize that this is how most jobs in computer science work, so I am glad that I got to experience how that changes my approach and process to writing code.