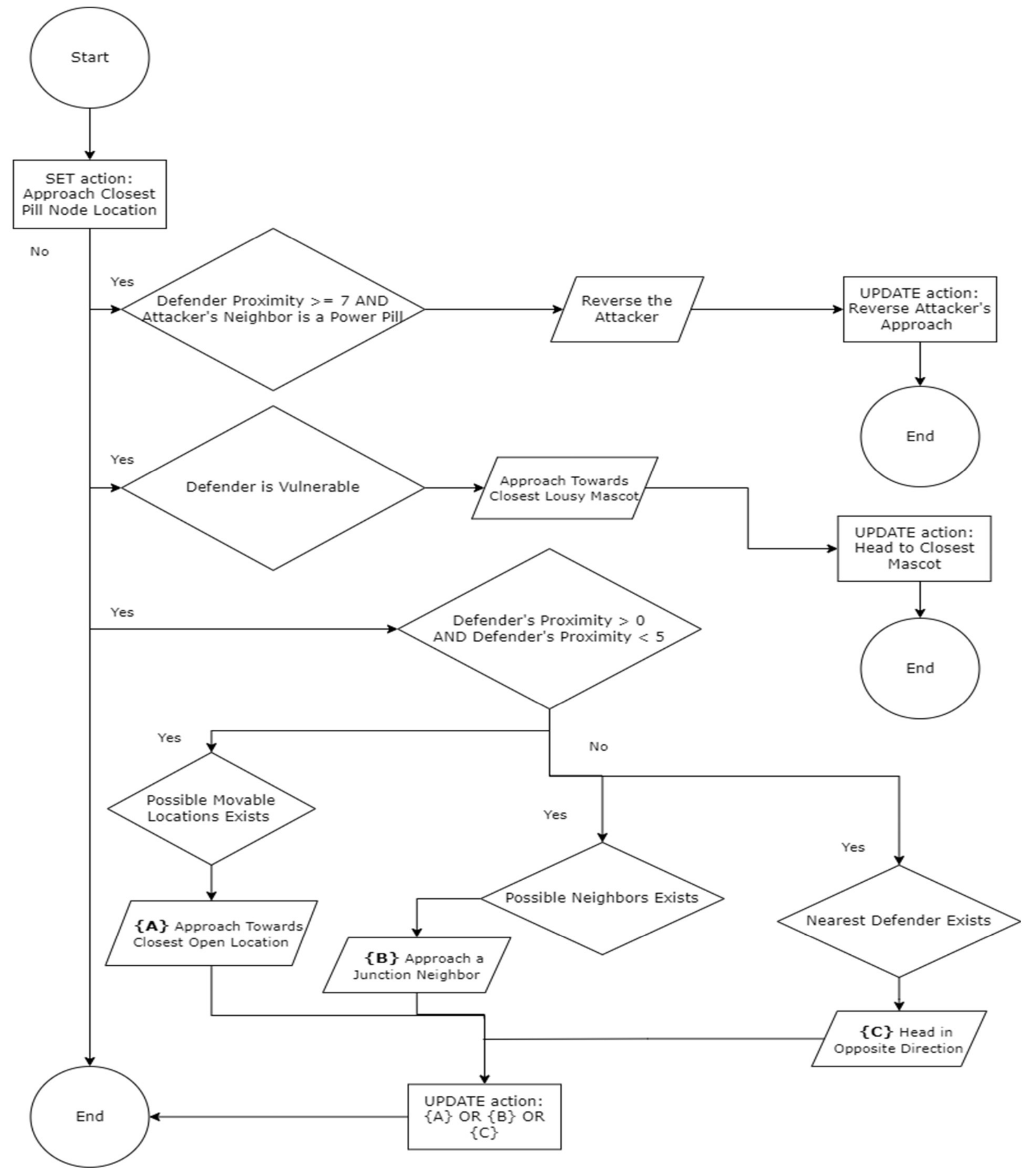
Design & Post-Mortem of Project 4

Mihir Patel

COP 3502

Design of Attacker's Action:


Start
SET action: Approach Closest Pill Node Location
No
Yes
Defender Proximity >= 7 AND Attacker's Neighbor is a Power Pill
Reverse the Attacker
UPDATE action: Reverse Attacker's Approach
End
Yes
Defender is Vulnerable
Approach Towards Closest Lousy Mascot
UPDATE action: Head to Closest Mascot
End
Yes
Defender's Proximity > 0 AND Defender's Proximity < 5
Yes
No
Possible Movable Locations Exists
Yes
Possible Neighbors Exists
Yes
Nearest Defender Exists
{A} Approach Towards Closest Open Location
{B} Approach a Junction Neighbor
{C} Head in Opposite Direction
UPDATE action: {A} OR {B} OR {C}
End

Attacker Behavior & Methods:

Upon the start of a game, the gator (games' Attacker object) next action is always to head towards a 'Pill' Node. However, depending on 'Power Pill' adjacency, Defenders' vulnerability and proximity correct the usual approach stated above. If a 'Power Pill' node is present as an adjacent neighbor, the gator gets instructed to reverse the direction. But, to trap the Defenders we instruct the gator to keep on reversing on the same position until some of the Defenders get in a vulnerability zone so that our gator can chomp on them easily. Upon numerous playthroughs with Test Agent - Visual, the best distance when the gator can proceed to 'Power Pill' is greater than or equal to 7.

After the gator eats a 'Power Pill,' the gator's approach changes, and it gets instructed to approach the nearest vulnerable Defender. When the vulnerability timer becomes 0 and there is no vulnerable Defender, the gator gets to continue its usual eat-the-pill instruction.

In a possible scenario where the gator is about to be caved in by the Defenders, it is instructed to head towards the nearest open location. Furthermore, if no nearest possible location is open, then the gator has two more options. The first option for the gator is to head towards the closest node which is a Junction Node. The reason for doing that has to do with Defenders' inability to enter the four Junction Nodes (or at least this is the case in the real-world Pac Man game). Second, if the current Node does not have a Junction Node as its neighbor, then the gator is told to head in the direction away from Defender/s.

Finally, three methods help achieve some of what has been said above. The first method is 'move_to_a_pill' which returns the location of the pill that the gator can approach. The second is 'check_weak_defender' which iterates over a list of Defender objects, and which one is vulnerable to gator's attack. Afterwards, the method returns that Defender's signature. The third is 'check_defender_attack_proximity' which returns the number of tiles a defender is from the gator's current position.

Identification of Successes & Failures:

Several strategies did not work the way they were planned out on the paper. Fortunately, enough pre-made functions were provided to get a score above 6400.

What Went Right – Fortunately, despite not having any experience with Pac Man, most of the strategies discussed during Dr Aman's Project 4 lecture were viable enough to code. For instance, in his lecture, he recommended starting simple and letting the gator move to 'Pill' Nodes. When a 'Power Pill' gets encountered, then let the gator approach the enemies. Furthermore, examining the experiment.txt and Example -Visual also flesh out an approach to a 'Power Pill' and how to behave near it and lure in the Defenders. Apart from the given tools, sometime got spent on developing the exit routes for the gator when the enemies are nearby. The three rules that were developed are (1) Priorities the Nodes in the shortest distance; (2) If any of the Neighboring Nodes are a Junction head to it; (3) Avoid the approach towards Defenders.

What Went Wrong – Not having a hands-on experience with Pac Man did not help come up with better strategies initially. The Lectures, Slack and YouTube were the best sources on “How-to-play” the game. Furthermore, many exit strategies sounded great on the paper, but developing them with given public methods is a challenge. For instance, one of the earliest strategies was to reach the junction and wait for n milliseconds. Or, if there are trailing enemies, check their future actions and adjust the gator's action appropriately. Likewise, it was hard to develop a quadrant-based pathing where the gator clears the four quadrants in a specific order. It may have been possible to develop a solution; however, that would require more familiarity with the classes, the methods, and more time to work on this project.

Reflection of the Project:

The project gives a good insight into the enemies, in general, getting instruction in video games. Even though this project's Actor Objects can get controlled using conditional statements, it would be interesting to see how pathing algorithms (such as A* or reinforcement learning) can get inserted and come up with different strategies. Since we get to keep all the files of this project, I may be able to incorporate such pathing algorithms into my project in the future. Aside from that, this project also provides a great introduction to Object-Oriented Programming, polymorphism, and abstractions. Likewise, the projects help us see that software development is not only about creating scripts in the src folder. Software development is a cohesive work of organizing various parts and giving tools to make something great.