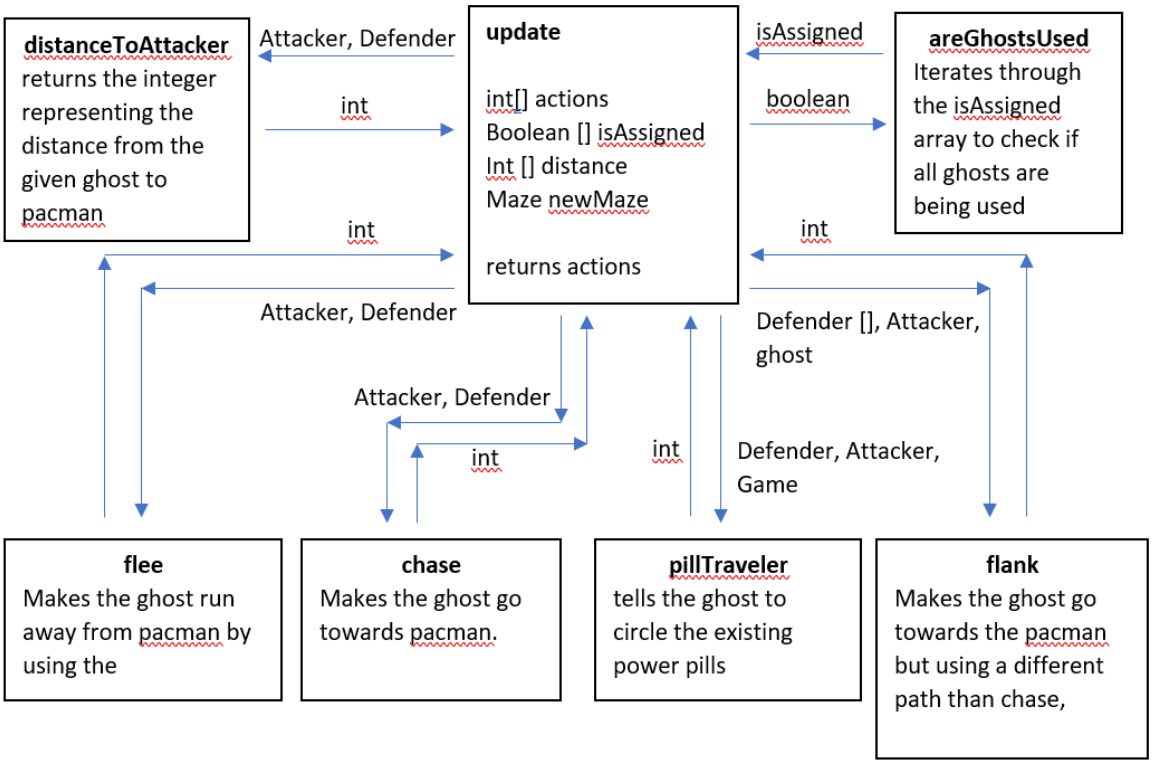
# Gator Raider Design/Post Mortem

Karan Aulakh (1163-4368)
Joshua Hartigan (9250-2966)
Caroline Hobson (5162-0769)
Cyan Perez (8359-6561)

# Diagram of Defenders Behaviors and Methods




*Methods Implemented*

public int [] update
public static int flee
public static int chase
public static int flank
Public static int pillTraveller
public static int distanceToAttacker
public static boolean areGhostsUsed

## Description of individual contributions each member of the team

Joshua Hartigan

- Coding/Design of Flee, Chase, and Flank Methods
- Post Mortem Diagrams

Caroline Hobson

- Coding/Design of Update method
- Debugging syntax/runtime errors
- Post-Mortem behavior descriptions

Karan Aulakh

- Coding/Design of Update method
- Post Mortem reflections
- Diagrams of Defender Behavior and Methods

Cyan Perez

- Coding/Design of Flank and Pill Traveller methods
- Debugging runtime errors
- Post Mortem

## Description of Each Individual Defender's Behavior and Strategy

1. Flee: This method was only implemented if the ghosts are vulnerable, and when they were vulnerable, it was used for every ghost. The .getNextDir() method was implemented on the defender using the attacker's location as the target, but false was inputted into the approach parameter. This returned the direction that the ghost should go to avoid the attacker. A while loop was then used to check the direction against the reverse direction of the ghost. If they were the same, then a different direction was selected. If they were not the same than that direction was returned as the action.

2. Chase : This ghost implemented the .getNextDir() method in order to find Pacman's location and assign it as its target node, and then also set the approach boolean to true so that the ghost indeed moves towards Pacman instead of away (as is implemented in the flee method). The location of the attacker was inputted into the .getNextDir() function as the target, which returned the direction in which the ghost should travel. A while loop was then created to ensure that the direction was not the reverse direction, and if it was a new direction was selected. The reverse direction was determined using the .getReverse() function.

3. Flank: The goal of the flank ghost was to also chase Pacman, but to avoid taking the same path as other ghosts. When the shortest path to Pacman was determined, each node along that path was checked to see if there was another defender there. If there was a defender along that path, then a different available direction was selected as the action. If there was no other ghosts between that ghost and the attacker, and that direction was not the reverse, that direction was returned as the action. This was done to avoid having the ghosts follow Pacman in a line, and instead attempt to cut him off. The reverse direction was not used as a possible direction.

4. Pill Traveller: This behavior method utilized an arraylist of nodes while there are power pills, found the nearest one and circled it in anticipation of Pacman. The distance to each power pill was determined using an integer array named 'distance' that held all the current distances of the ghost to the power pills. Then the method iterated through the array and found the closest distance to use. As each pill was eaten, the method removed it from possible targets and found the next closest pill to stalk. If Pacman ate all the pills in the map, this ghost defaulted to a flank behavior.

## Description of Overall Team Behavior and Strategy

The defender team aimed to both proactively attempt to corner Pacman while also implementing strategic preventative measures to ensure he could not score substantially more points. For instance, the chase ghost's sole job was to find the quickest path to its target: Pacman. This role was given to the ghost that was closest to Pacman's current location. The second closest ghost to Pacman was assigned as the pill traveller. This way, even though this ghost would probably not have enough time to get in between Pacman and any pills he was close to, it could make sure the next available power pill Pacman may want to use was guarded. Finally, the last two ghosts were assigned to flank, which found the shortest path to Pacman, given there weren't any ghosts already on that path. If at any point Pacman ate a power pill, all ghosts were assigned flee. This behavior caused all ghosts to immediately begin maximizing their distance from Pacman. This assignment only lasts as long as the ghosts are vulnerable. Once vulnerability ends, the ghosts resume the appropriate designations.

## Evaluation of Performance of Individual Defenders and Team

Improvements could be made to all of the methods in order to make them perform even better than they already do. In the chase, flank, and flee methods if the reverse direction was found to be the fastest route, a random direction is chosen. Instead of choosing a random direction, the second shortest path could have been chosen. Similarly, in the flank method if another ghost was in the shortest path, again a random direction was chosen rather than the second shortest direction. For the pill traveler ghost, if Pacman does not go completely over the power pill when he eats it (just goes within the range of four spaces and turns around), then the pill traveler ghost does not know to leave that pill for another and continues to circle that location even though there is no longer a power pill there. The ghosts could have caught Pacman faster if these changes were implemented, however, they still performed well enough to be within the goal. As a team they worked very well by assigning action methods based on the distance away from the attacker, rather than permanently assigning ghosts a specific strategy. This allows for situations like if Pacman gets close to the location of a power pill, the pill traveler ghost can switch to chase and attack rather than continuing to circle the power pill.

## Identifications of Successes and Failures

Upon reflection of the project's performance, it was agreed that there were not only efficient techniques that were invoked, but also there existed several limitations to either what could be done given the base code or given the students' current programming abilities. For instance, a failure of the team was when it was attempted to create a density behavior that would allow a ghost to circulate through a quadrant of the map that contained the most pills so that when Pacman attempted to go collect more points, there would be a ghost waiting for him. However, it was determined to be too time consuming and inefficient to code a way to not only dissect the map into quadrants, but also count the pills in that quadrant and even retrieve the appropriate x and y coordinates of the map nodes. Once this was realized, the team ultimately understood the power of how the current methods were being utilized and focused more attention on which methods were being assigned to which ghosts. Through this new mentality, the group was also able to design a new behavior named 'pill traveller' that did not focus on pinning down Pacman, but instead focused on preventing Pacman from getting more power pills. A big issue in the game for the defenders was that Pacman would hover next to power pills until they got close, then eat the pill and then eat all the ghosts in his proximity. Thus, it was inefficient to have all ghosts head to his vicinity. The thought process behind the pill traveler method was definitely a success of our project.

## Team reflection

At first, the project was overwhelming. Being thrown into a code base with premade interfaces and objects that cannot be altered was initially difficult and confusing. Once the basics of the code base were understood, it seemed less daunting to actually begin developing the controller. There was a lot of conceptual misunderstandings going into the project that hindered the team's progress. One example was when trying to prevent any of the defenders going in reverse. Although it was not allowed to implement explicit code telling the ghosts to reverse, the base code automatically forces them to reverse at random intervals. However the team was not aware of this and it took a lot of time to understand that it was not the student controller causing the issue but the base code and that it was an issue that we did not have to counteract. Another example is when determining whether each ghost had to have their own unique behavior at all times, or if there were only four behaviors required and they just had to all be used at one point or another. Due to the group's ability to aid each other in the facilitation of troubleshooting bugs and runtime exceptions, the main goal was accomplished: keep Pacman from achieving the example average.

# Individual Reflection

Caroline

Having taken the majority of this class before and having personal experience, I have noticed the structure of projects vary depending on the professor. There is no right or wrong way for developing a project, and although starting this project was sort of a struggle, it definitely was not the worst project I have completed. I get overwhelmed very easily, so the major obstacle for me was understanding the PDF given. At first glance there were a lot of instructions, descriptions, and requirements, so it took me a some time to read through the entire pdf so that I could fully understand it,. Fortunately interfaces were pretty easy to understand, and I was very familiar with objects and classes, so I was able to assist my group members with any conceptual questions or easy debugging. For example when we called multiple methods inside each other, it was difficult to determine which method returns what when you start to have errors. I was able to translate exactly what method was calling what, as well as help them find the correct methods that they needed to complete a behavior. Overall I think our group had a good mix of strengths that complement the weaknesses, which is why we were able to work so well together.

Cyan

One strength that I found when approaching this project was my ability to visualize all the repercussions of the code I was writing. If I wrote a for loop to iterate through an array list that may lose indexes, I created a while loop to ensure it would only initiate when there were values to go through. However, a weakness I encountered was my lack of understanding concerning the scopes of all the different arrays and variables. There were many times when I would attempt to use a certain int array or long that I was certain I could call, and yet that evil red squiggly line would still appear. I combatted this by creating a 'cheat sheet' of scopes for different data types and quizzed myself throughout the project. This not only helped me get through this project, but also proved to be a useful tool in studying for our upcoming final. As a team, I believe we worked well together. We have unique dynamics to our personalities that work together quite well. I am very motivated to complete tasks and so kept the group on task and meeting regularly. Caroline has a well of previous programming experience. Josh can debug anything you throw at him. And Karan has the ability to think like a machine.

Josh

The most difficult part of the project for me was understanding the code given to us. It took lots of reading between the pdf of the project and reading through the code to piece together how it all interacted. The biggest hurdle was fully understanding how the interfaces were implemented and how to accesses methods from outside the student controller within our code. After I understood how the different interfaces and classes worked together, I was able to then begin designing the different action methods. When designing the action methods, I tried to utilize the given methods as much as possible in order to make the action methods as short and efficient as possible. I also reused the same logic progression between the different behavior methods as much as possible in order to make my life easier. I learned a lot from this project about how I can organize my code in the future when creating more complicated projects and about the usefulness of tools such as interfaces, abstract methods, and subclasses to make my code more organised and easier to understand. Overall, I am satisfied with our performance on the project and the quantity of material that I've learned from it.

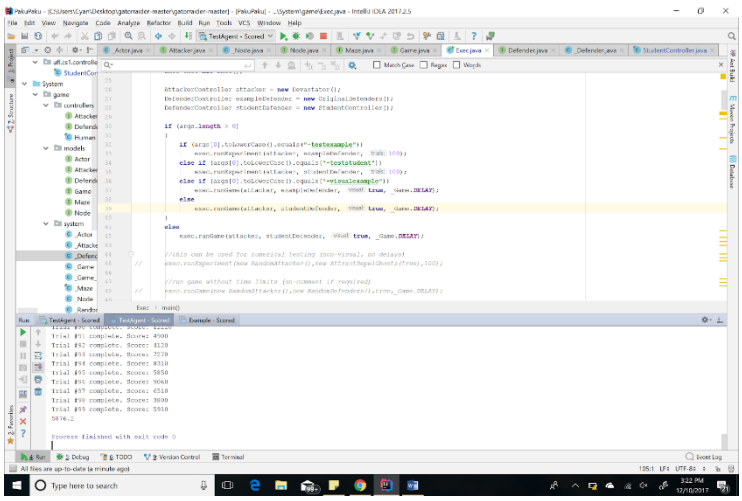
Karan

When I initially started this project it seemed overwhelming and hard to start. However, once I went through the code and started understanding how it all works, the project started to look a lot more reasonable. Setting up the project this way was a good idea because it forced us to work with a high level of code and understand the way an entire project looks. Even though, we didn't make the whole project, we were able to work with the various methods and classes and learn from it. The project itself wasn't

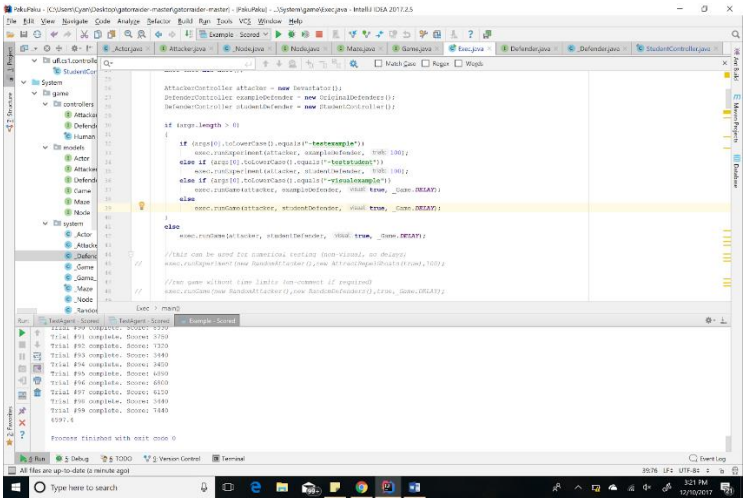
that hard since we didn't write that many lines of code and barely got any bugs we had to work with. Another part of the project I liked is the openness and room for thinking in it. For example, the last projects were very detailed in the setup and the output of the projects. Almost all of us had the same methods and setup for the projects. However in this project, a lot of thinking had to be done by us. We weren't given any output or specific behaviors we had to implement which required us to problem solve and think about different ways to tackle this problem. The only thing that was annoying was the bugs in the code, especially the fact that some laptops weren't compatible with running the right scores. This caused some troubles with being able to work on the project on my own.

## Screenshots

*Student Controller tested at 100 Rounds*



*The example tested at 100 Rounds*



**The Java version and platform your code was run on for the screenshots**
Java 1.8.0_144
IntelliJ