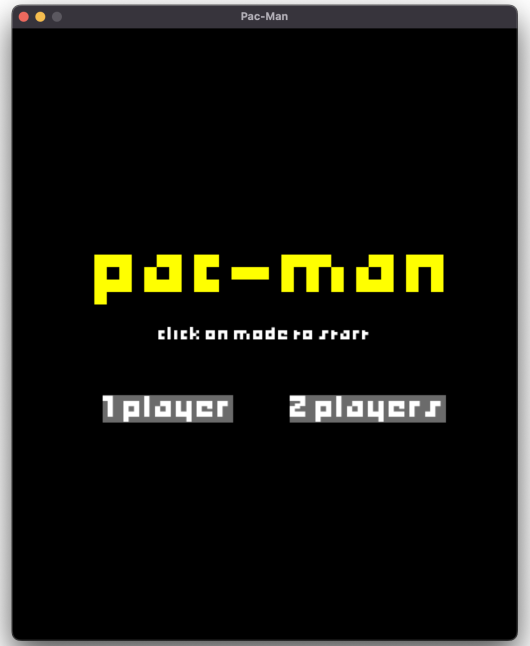
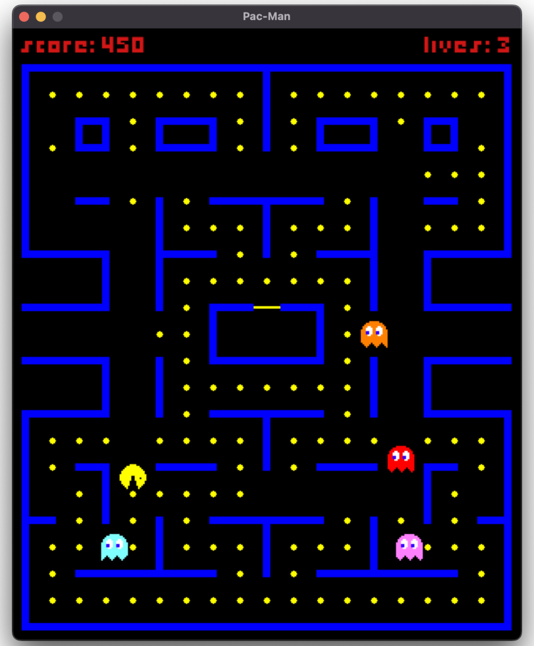
Project: A multiplayer remake of the original Pac-Man arcade game
Team name: Pac-Man
Group members: Qing Zhang, Tianyi Zhang, Neil Powers
Play it yourself: https://github.com/theQuarkBot/PacMan

You've probably played Pac-Man before, but have you ever played it with a second player – and were they a ghost? Just like the classic one-player game, you explore the retro maze, collecting pellets and evading the sly ghosts. If they catch you three times, you're done. Of course, eating a super pellet reverses the roles, letting you chase the ghosts towards the end of the earth. Once all the pellets are eaten, the game is over and you can see your score. Unlike the classic game however, another player can assume the role of a ghost, making the game that much more confounding. Get the highest score you can…but don't get caught!





The game was built mainly using the Pygame module and the built-in threading class. Whenever the game advances to a new frame (60 times per second), the game first displays the board and each character, then it captures the current key-presses to send to the characters. Each character has a thread which accepts the key-presses, and uses them to calculate the next move. If a bot controls the character, then it simulates its own keypresses. The game only advances to the next frame if all characters finish calculating their next move.

Refined Report:

Minimum and Maximum Deliverable:

- Updated: At a minimum, we will have 2 multithreaded objects, one is the board/screen and the other is a pacman moving across it.
- This is our goal(Maximum deliverable), it is a normal Pac-Man game with multi-threaded ghosts and players.
    - Similar to the default pacman game: a single player with a couple ghosts
    - Game state is in one thread, the game board stored as a 2d array.
    - Each player/ghost is running on its own thread. They get the next move and send it to the game.
    - Doesn't necessarily have to have a score; just a displayed board with threading ghosts and a threaded player.
- At best, it is a multiplayer Pac-Man where users control the ghosts and Pac-Man(s). There will be lives and a point based system.
    - Similar to single-player, game state is on its own thread. Each player/ghost will be running on its own process/thread, sending moves to the game state thread and receiving display information (if necessary) if it is a player.

Problem? How to balance the multiplayer pac-man game so that the pac-man team has a similar chance of winning as ghost team

- Two modes were created, one with a user controlled and one without
- We have decided to make one user controlled ghost with multiple random movement ghosts
- Added super pellets, only 4 such that Pacman player would have to strategically plan when to access them

Design Decision:

On our first meet up day, we all already had in mind that we would be doing a pac-man game and that the minimum deliverable was to simply make a pac-man game. But we thought about what if we made it multiplayer, such that there was a pac-man team and a ghost team. There was an issue here because if we had 4 ghosts and pac-man, then the game would be almost nonexistent since the ghosts could just corner pac-man. This led us to ponder about what possible solutions there are in order to balance this game. The ideas that we were able to come up with are: equal number of pac-man as ghosts, more power pellets, make pac-man faster than the ghosts, and to make the game a point-based system where whichever team finishes with the most points wins. Our decision was to simply keep all of these in our heads, and once we get to this step, we will just test them all. Since we need to find the solution that balances the game the best, we are unable to make an accurate decision without testing them first.

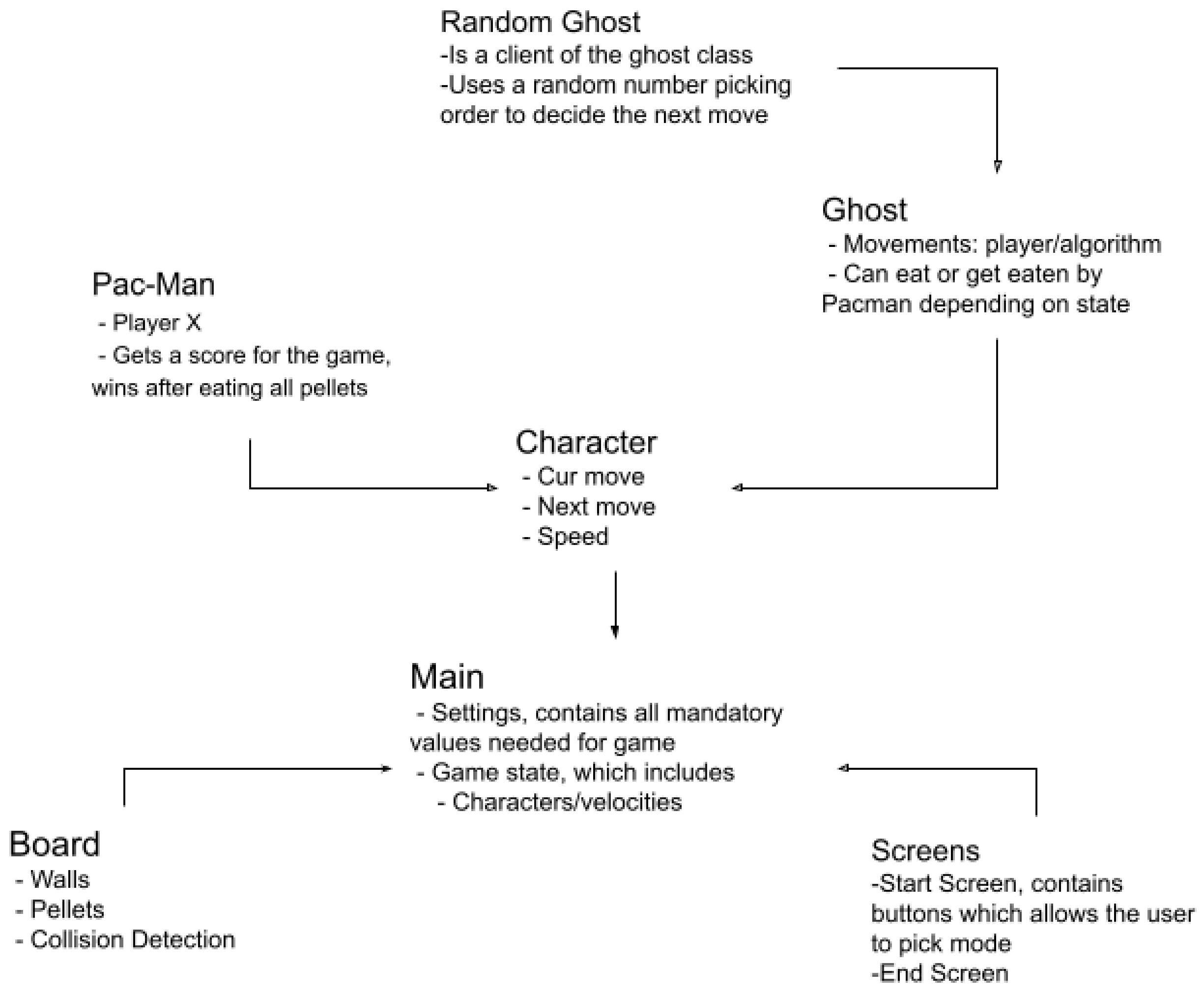
What we have done:

1. Brainstormed structures and implementation
2. Got pygame to work, looked through some pygame methods
3. Made basic map, then added updated neighbor detecting map
4. Made pacman move with user control and change image when moving.
5. Made a main which controls the running of the program
6. Made pacman and ghosts work with multithreading
7. Combine the map and characters
8. Made characters check for walls before moving
9. Make a functional simple game (Moving Pacman eating pellets)
10. Added random moving ghosts
11. Different states added for ghosts, when super pellet is eaten, ghosts slow down and can be eaten by pacman
12. Flashing for last 2 seconds when weakness is about to end
13. A score board and lives were added, how well player does depends on their final score
14. Added start and end screen
15. With buttons to choose player mode
16. Cleaned up character classes, use inheritance and abstract class
17. Music is added

Work Division/Interface Management:

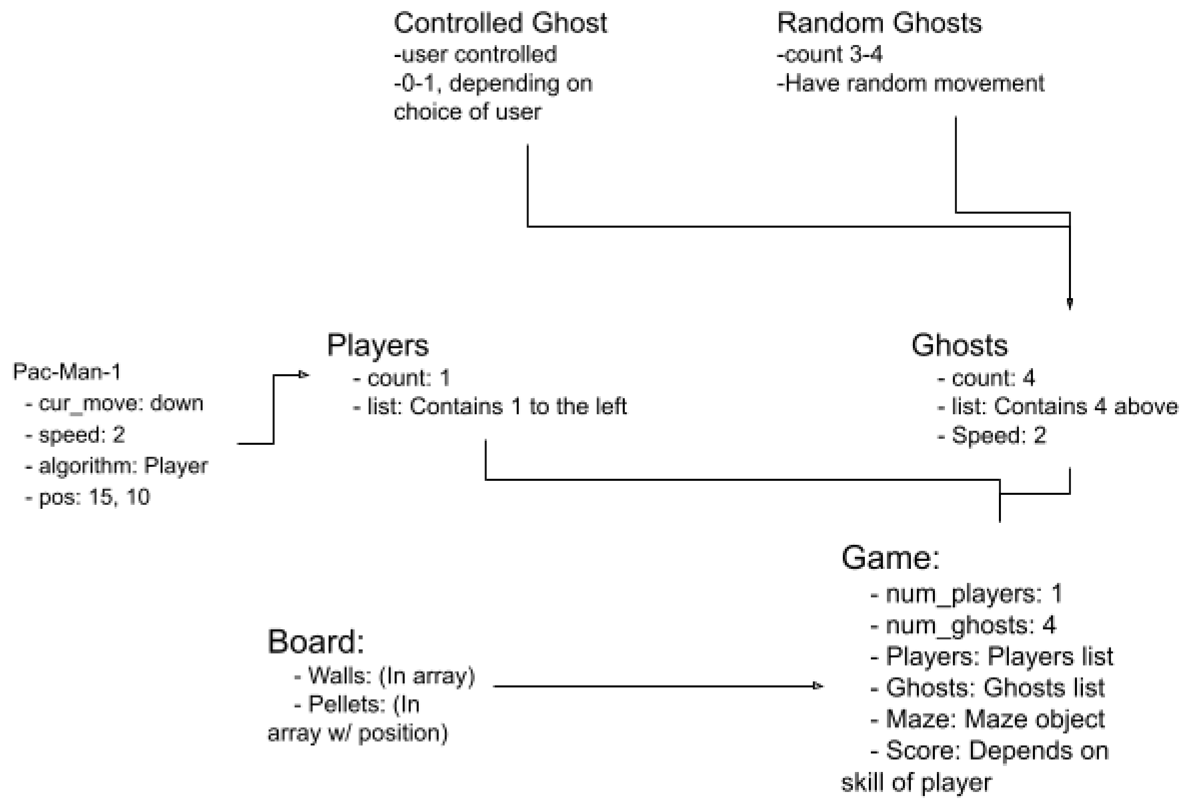
1. There will be no division of work. Everyone will be involved in every part of the project, such that no one is out of the loop at any point.
2. Updated: Divide the work naturally, each person finds a part they will work on when not meeting.

Class Diagram:


Random Ghost
-Is a client of the ghost class
-Uses a random number picking order to decide the next move
Ghost
- Movements: player/algorithm
- Can eat or get eaten by Pacman depending on state
Pac-Man
- Player X
- Gets a score for the game, wins after eating all pellets
Character
- Cur move
- Next move
- Speed
Main
- Settings, contains all mandatory values needed for game
- Game state, which includes
- Characters/velocities
Board
- Walls
- Pellets
- Collision Detection
Screens
-Start Screen, contains buttons which allows the user to pick mode
-End Screen

The Board Class contains instances of the characters and the Maze board.
The Character class is a parent class of the Ghost and Pacman classes. The Pacman class is always controlled by a player, while the Ghost class can be controlled by a computer or a player.

An example object diagram after



The board class is going to store all of the locations of characters and pellets. It doesn't really need to know what the characters are, it will just call corresponding methods when two characters meet, when pellet meets a character, etc. The character's class will handle it accordingly. For example, if the characters are two ghosts, nothing happens. The characters also store their own speed and modes and a copy of their coordinates. They will signal the board constantly when they move.

All diagrams will remain the same, since we implemented our program the way we planned it out to be.

Analysis of Outcome:

We did very well on this project, almost perfectly fulfilling our maximum outcome. We were able to make a functional Pacman game that could support 2 players on the same device at a time. The weekly meetings that we had really helped with our progress because during that meeting we would set goals on what we would want to accomplish throughout the week. With this we were always on track and able to avoid situations where we were falling behind schedule. It played a large role in our success as a team, keeping us engaged with each other and the project. The only element that we were unable to incorporate into our project was making the game available on a server, such that people from different devices could play. This was due to us spending a lot of time on the logistics of the game. The detailed logistics of the game were figured out and implemented into the game in the end. Even if we were able to make the game playable from multiple devices, we would have to spend a lot of time deciding how to assign what roles to which user unless it was made random. The concurrency that we implemented in this project was by making each character its own thread and nothing else because we felt that although there were multiple opportunities to do threading, it would be extremely unnecessary, so we only implemented threading in the area that it would make the most sense. Otherwise our project was a great success in all other aspects with the game fully functioning and very playable.

Reflection on the Design:

The best decision was drawing out the diagrams because when we did start programming, we didn't have to spend a lot of time trying to figure out what goes where and we could instead just smoothly implement everything without having to constantly communicate with our teammates to make sure that everything each person wants to do is aligned correctly. The part of our design that did not go the best was the development plan and trying to plan out when we should have each aspect done by. This was mostly due to some classes being much more difficult to code than others, while others took almost no time at all. That might also be a fault of ours for not completely understanding the difficulty of each portion, but we have all learned from this. Because as we started working on the project, what we wanted to do each week was more and more reasonable and there was no scenario where we didn't have something to do or there was too much to do. Also simply having the design allowed us to keep track of what we had to do and what was done so there was no need to meet up over and over again discussing what had to be done.

Division of Work:

The work division went a lot smoother than expected with the use of github, such that as long as we had each teammate understand what part we were working on, there was almost no collision in changing the code. We did not assign work to each person, it was more so each person choosing what they wanted to do for that week and then doing it.

Bug Report:

The code to update each character is fairly complex and not at all easy to debug, and when you wrap it all in a layer of threading, it becomes that much harder. When we were first working out the movement of the characters, this became especially apparent when we were converting a working non-threading pacman to a threading pacman. If there was even the tiniest bug, it would cause the program to halt, since a thread would crash causing all the other threads to hang. One particular bug prevented the pacman from moving whatsoever.

We initially thought it was caused by it failing to read inputs, since that is the part we changed. However, after adding numerous print statements throughout the program, we eventually discovered that the characters' hitboxes were two pixels too large when they spawned. Since the hitboxes were *inside* the walls, it was impossible to move them. We eventually discovered that the source of this bug was in scaling images. Each grid in the board was 30x30, while the pacman was scaled from 16x16 to 30x30 – or at least that's what we thought. After transitioning to a threaded version, they were actually scaled to 32x32. It took much trial and error, but we eventually found this bug and rescaled the board to a 32x32 grid. This bug took around 30 minutes to find simply because of the misdirection caused by working on something else completely unrelated. To find it even faster, it probably would have

Overview of Code:
Bin: Contains images that were used to represent the ghosts and pacman, Also contains fonts used for our screens.
board.py: Contains the Board class. When called, creates a pygame screen with our board/map on it.
main.py: Is the center that links all other files together. Creates all objects that are needed for the game and has the while loop that runs until the game is over.
pacman_class.py: Contains the Pacman, Ghost, and RandomGhost classes. All functionalities of each object are declared here with all movement and collision parameters(with walls) are here.
screens.py: Contains buttons for the start screen, the start screen and the gameover screen. Used by main.py for us to know which version of the game the user wants to play.
settings.py: Contains constant values that every other class uses, such as game states, pygame controls, starting locations of pacman and ghosts, etc.
thread_safe_classes.py: Contains two classes that are used by the Pacman, Ghost and RandomGhost classes in order for us to thread the objects safely without race conditions or deadlocks.

Instructions on how to run:

1. Have all files and bin in one folder
2. Make sure you are able to run pygame on your device
3. Call python3 main.py
4. Pick if 1 player or 2 player mode wants to play, if 2 find a friend.
5. Pacman moves with arrow keys and Ghost moves with WASD
6. Play the game!

Packages Used:

We used some aspects of Pygame to help us, such as the creation of the screen, collision of objects and creating the objects. On the Halligan servers, pygame does not work because no video device is able to be found.

We did download some fonts and music that are stored in our bin in order to have some sort of retro text on our screens.