# Research

To get a better understanding of how the implemented game should turn out, research was conducted into the original product, Pacman. Pacman is a game in which a player navigates through a maze collecting all the pellets (points) while avoiding the ghosts (enemies) that chase the after the player. The game ends once all the pellets are collected or when the player loses all of their lives.

Although. through the research conducted, most aspects of the game were identified as ‘easy logic’ and standard, the research did find credible information. Such information was under the turning logic and the artificial intelligence algorithms for the ghosts.

## Turning Logic

When the player comes across an intersection in the maze, the player should be able to change directions in two dimensions. How the game decides when to make a turn is to first read the last user directional user input, store that information and make the appropriate turn when the intersection is encountered. The user does not need to hold the directional key down or make the decision at a ‘pixel perfect’ moment.

## Artificial Intelligence

Ghosts make decisions on where to advance by switching between states. These states can be labelled as scatter and chase. An additional state (frightened) exists, however this mode was not accounted for in the development of the game. During scatter mode, the ghosts try to reach specific locations outside of the maze whereas during chase mode, the ghosts ‘chase’ after the player. These states transition between each other after fixed periods of time. The transitioning times also vary as the user advances in levels.

Ghosts cannot make reverse directions except under an exception. This exception is when the ghosts transitions from only scatter or chase modes, at which point the ghosts must reverse. The ghosts do not reverse when transitioning from the frightened state, however.

Ghosts cannot at any time during gameplay stop moving.

Another important note is that the ghost AI do not have sophisticated methods of chasing after the player. These methods include the communication between ghosts, no use of path finding, and machine learning.

Because path finding is not utilized, ghosts need to make the decision on which turn to make at an intersection through another means. This is done by looking at all the tiles around the intersection tile where a turn can be made. Upon arriving at the intersection, the ghost makes the turn onto the tile that is the closest to the target position via a straight line. This means that a ghost could make a turn into a tile that may be closest to the target position, however may travel a longer path.

Each ghost also carry unique features however these features were not considered through the development of the game.

# Design

Upon development, 6 screens were implemented, however only 3 had content. These screens were the welcoming screen, the game mode selection screen, and the gameplay screen.

## The Welcome Screen

What was required of the welcome screen was to prompt the user to select a game mode through a keyboard. Instead the options were moved to a seperate screen and another navigation screen was presented which included navigation into an achievements screen, store screen, settings screen, as well as the game mode selections screen.

The navigation consists of options in the form of Rectangle classes and Text classes. Upon navigating the options through keyboard inputs, the colour of a rectangle changes to indicate to the user which screen they will transition to upon pressing the “Enter’ key. This change is conducted through the highlightUp() and highlightDown() methods under the Environment class. When highlightUp/Down() is called upon, the rectangle directly above/below changes colour to PINK, while the rectangle that was PINK before reverts to AZURE. No new rectangle is formed in this operation.

The Rectangles and Texts are constructed through the makeOptions() method under the Environment class. The method takes for inputs a String array, desired width, desired height, desired spacing, fill colour, stroke colour, and stroke width. The String array contains the list of desired option names. This method creates the rectangles and texts and positions them in evenly about the centre of the screen.

These rectangle and text classes were used as placeholders for when the proper graphics were made available.

## The Mode Selection Screen

Upon selecting the “Play!” option from the welcome screen, the player is navigated to the game mode selection screen. Here, the player can decide how many players there will be (from one to three) or navigate back to the welcome screen. The buttons were made reusing the makeOptions() method and the navigation is conducted through the reuse of the highlightUp/Down() methods.

When the user selects from the number of players options, the user is sent to the gameplay screen.

## The Gameplay Screen

Upon entering the gameplay screen, a countdown is displayed. The countdown runs with transitioning Text from “Ready?” -> 3 -> 2 -> 1 -> “GO!”. During this time, the 2 minute timer is on hold, the player(s) cannot make movements, and the AI are all stationary.

Noting that the handle method runs at a rate of 60hz under the AnimationTimer() class, the time that passes can be generated by counting and updating the frames.

Every second, the countdown text is updated through the doCountdown() method under the Timer class. This updates the String input for the Text class that gets used as the input for Group.getChildren().add(Text) method. Because by adding the same Text generates errors, the text is first removed using the Group.getChildren().remove(Text) method before updating.

After the countdown finishes, player(s) can provide inputs to control the character(s) and the AI also move according to the chase algorithm.

If there is one player, the player can control the appropriate character using the arrow keys. A second player can control a chaser using ‘WASD’ and a third player can control another chaser using ‘IJKL’.

The timer runs down from 2 minutes updated through the doGameTime() method under the Timer class. The visuals of the timer uses the same principles as the countdown. Upon collecting the pellets, the score increments and the pellet disappears from the screen. The logic for this is further discussed under Development Problems.

When the AI collides with the player, the player is sent to the reset position, the timer is put on hold and the countdown timer restarts. The AI and additional players are also meant to return to reset locations, however this method was not implemented. This method would consist of using the Character.setXPos(), Character.setYPos(), Character.Direction being called upon along with the reset method (Player.resetALL(boolean, int)) of the player.

The player should have a total of 3 lives but the method for ending the game after 3 lives was also not implemented. This method would consist of an “if” statement that checks the number of lives the player has. Upon reaching zero, the game should end.

Other inputs include the “P” key which toggles the pause of the game by changing the state of the game. Updates to the game can only be made under a certain game state and by exiting from the correct game state, the game is put on pause. The “ESCAPE” key prompts the user to confirm whether to quit the game or not. This also changes the state of the game effectively pausing it. The “PAGE DOWN” key instantly ends the game. This is done by reaching the outer time bounds while also setting the timer to “0:00”. The user returns to the welcome screen upon pressing the “ENTER” key.

The player can play again by selecting the appropriate options of the navigation screens. Upon appearing at the gameplay screen, the position of the player, the score, the pellets, the countdown and timer are all reset. What is not reset is the positions and states of the AI and additional players.

# Development Problems

Upon development, due to inefficient and non-flexible methods, some features were difficult to implement. This include the problems with pellet collection and using a unit map.

## Pellet Collection

## Unit Map vs Pixel-by-Pixel Map