<InfPALS/>

Big Project

Session 1

[These slides have been updated on 15/3/2024]
[They include corrections to the starter template files]

Project Overview - Pac-Man in Pygame

- Session 1
  - Game loop
  - Drawing the board
  - Drawing the player
  - Player movement
- Session 2
  - Wall collisions
- Session 3
  - Implementing ghosts

Starter Files

GitHub

Corrections to starter files. Apply these corrections upon unzipping the files

1. Line 1. Replace import statement with import pygame, random,
2. Line 18 “BOARD_COUNTER=5” delete the line
3. Line 79 “self.previous_time = 0” delete line.
4. Line 89 “def actions(self, board)” change to def actions(self, time_delta):

Understanding the Starter Files

- There are some numbers and functions given:
  - Constants:
    - Define fixed values like screen size, tile dimensions, and game settings.
  - Board:
    - Represents the maze layout using a 2D array (list of lists) with values indicating tiles (empty space, pellets, etc.).
  - Player class:
    - Encapsulates player properties (position, direction, speed, score) and methods for movement, drawing, and interaction with the board.
  - to_tile(x, y):
    - Converts a screen position (x, y) to its corresponding tile coordinates within the board array.
  - draw_board(screen, show_powerup):
    - Renders the maze layout on the screen based on the board data, potentially considering power-up visibility.
  - main():
    - The main loop that keeps the game running, handling user input, updating objects, rendering visuals, and controlling the frame rate.

Game loop

- At the bottom of the main function, implement the game loop:

- 1. WHILE the game is running
  - a. CHECK for any events (quitting the game)
  - b. UPDATE the positions and actions of all objects in the game
  - c. CLEAR the screen
  - d. DRAW the game elements (board, player, ghosts, score, etc.)
  - e. DISPLAY the updated screen
  - f. CALCULATE the time difference (in seconds) since the last frame
  - g. CONTROL the frame rate (e.g., aim for 60 frames per second)

Game loop – Hint

- **pygame.event.get(): Li**st of events that have occurred (like key presses).
- Each element of pygame.event.get() has an attribute .type indicating its type. We want QUIT type
- **pygame.quit():** Quits the game.
- **player.move():** Updates the player's position based on user input.
- **player.actions(time_delta):** Handles player interactions with the board (pellets, power-ups).
- **screen.fill():** Fills the screen with a background colour.
- **draw_board(screen, show_powerup):** Renders the maze layout on the screen. (This might be provided)
- **player.draw(screen):** Draws the player sprite on the screen.
- **pygame.display.update():** Updates the display to show the rendered graphics.
- **CLOCK.tick(fps):** controls frames per second and returns time delta in **miliseoconds**

Draw the player

- In Player.draw()
  - 1. Draw the player's image onto the screen at its current position

Draw the player – Hint

- **screen.blit(image, rect):** Draws an image (player sprite) onto the screen at a specified position (rect).

Player movement

- In Player.move():
  - 1. Get user input (WASD keys)
  - 2. SET direction based on which keys are currently pressed
  - 3. IF W is pressed
    - SET direction to 0

Repeat for other keys.
Remember directions are 0=Left, 1=Right, 2=Up, 3=down

Player movement – Continued

- In Player.move():
  - 5. UPDATE player's direction based on currentDirection
  - 6. IF currentDirection is Left
    - a. DECREASE player's X position by speed
  - ELSE IF currentDirection is Right
    - a. INCREASE player's X position by speed
  - ELSE IF currentDirection is Up
    - a. DECREASE player's Y position by speed
  - ELSE IF currentDirection is Down
    - a. INCREASE player's Y position by speed
  - 7. Handle wrapping around the maze (check if going off one side, appear on the other)

Player movement – Hint

- **player.direction:** Stores the player's current direction (left, right, up, down).
- **player.speed:** Defines the speed at which the player moves per frame.
- **player.rect:** Represents the player's position and size as a rectangle.
- **Mydict = pygame.key.get_pressed():** Returns a dictionary indicating which keys are currently pressed.
- Mydict[K_c] returns true if key c is pressed (view keycodes at top of file)

Player actions

- In Player.actions():
- GET current tile
- IF player is on board
  - 1. IF player is on a pellet
    - b. REMOVE pellet from the maze
    - INCREASE score by 10
  - 2. IF power pellet eaten
    - a. SET player to powered-up state
    - b. START power-up timer
    - INCREASE score by 50
    - //frighten ghosts
- **IF powered up**
  - **IF timer still running**
    - DECREASE the timer
  - **ELSE**
    - DISABLE power up
    - //unfrighten ghosts

Player actions – Hint

- **board[y][x]:** Accesses the value (empty space, pellet, power-up) at a specific tile on the board (y = row, x = column).
- **player.score:** Stores the player's current score.
- **player.powered_up (boolean):** Indicates if the player is currently in a powered-up state.
- **time.time():** Gets the current system time in seconds (used for power-up timers).
- Look at tile_x to know if player is on board
- time_delta = CLOCK.tick(60) / 1000

Big Project

Session 2

Pacman movement in the maze

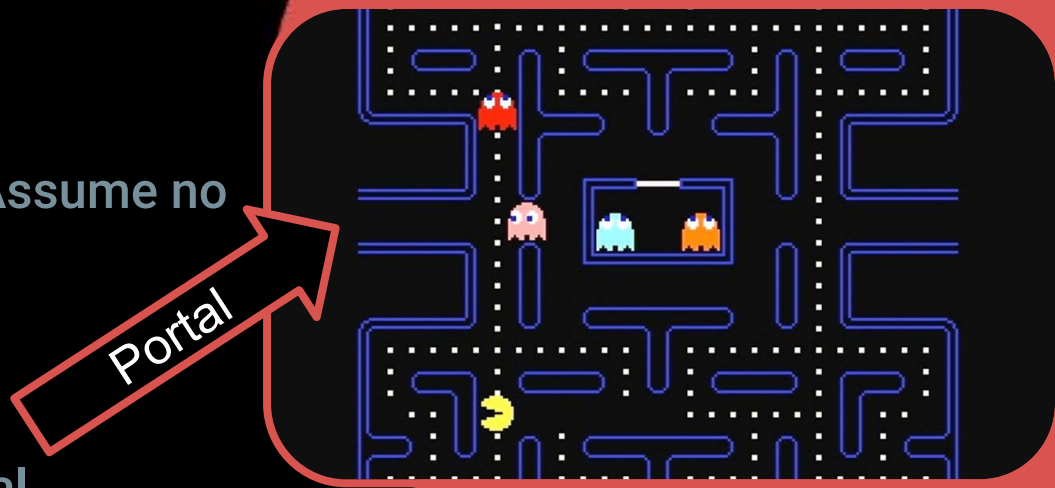
- Checking adjacent tiles for walls
- Colliding with a wall
- Moving in a straight line and at junctions
- Moving the rect (based on last week)
- Moving around the edge of the maze

wall_check(checking for wall left, right, up and down)

Function that takes centerx and centery of pacman and looks_ahead several pixels

```
def wall_check(x, y, look_ahead, can_enter_gate):
```

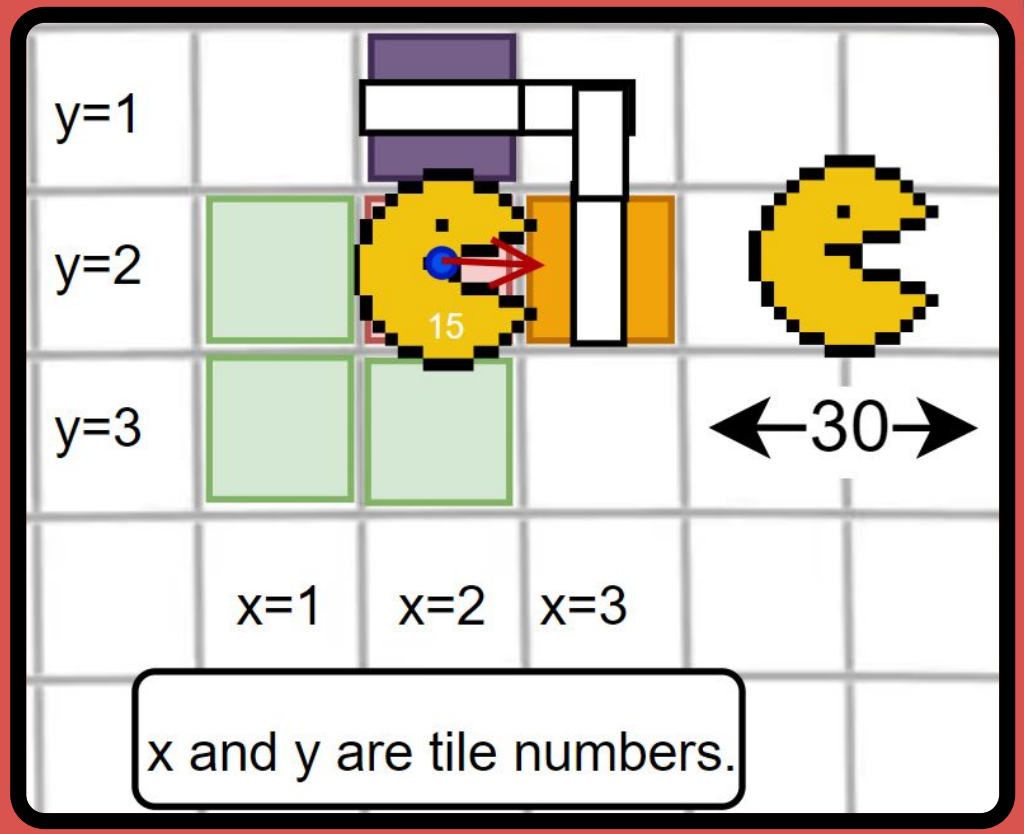
1. GET tile_x and tile_y of pacman is in using to_tile()
2. IF tile is within left and right limits of board
   a. SET walls to LIST of FALSE FALSE FALSE FALSE (Assume no walls)
   b. CHECK WALLS (later slide)
3. ELSE
   a. // We are going around the board through the portal
   b. SET walls to left and right is FALSE and up and down is TRUE
4. RETURN walls



Hint

- Variables x and y are the centerx and centery of pacman
- When tile_x is 0,1,2, .. TILES_WIDE-1, x is on board
- walls is a list of 4 booleans

CHECK WALLS



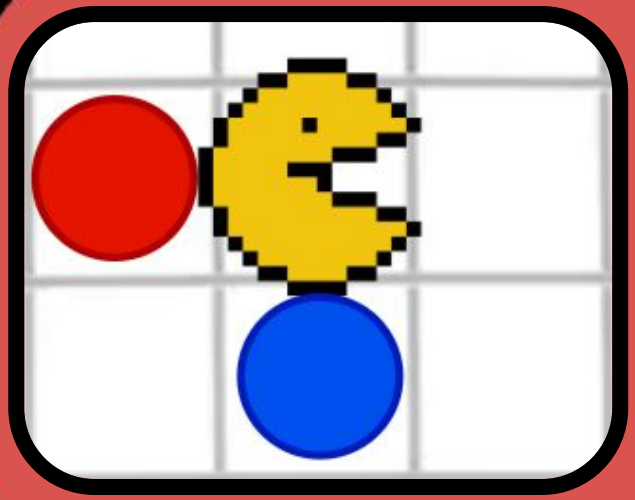
What value should look_ahead be?

(After SET walls to FALSE,FALSE,FALSE, FALSE)

1. GET tile number of tile look_ahead pixels Left, Right, Up and Down using to_tile()
2. IF can enter ghost house gate (gate=9)
   a. IF Left tile in board contains (3,..8)
   b. SET wall[0] to TRUE
   c. (Add contains conditions for Right - wall[1], Up - wall[2] Down - wall[3])
3. ELSE
   a. IF Left tile in board contains (3...9)
   b. SET wall[0] to TRUE
   c. (Add for Right - wall[1], Up - wall[2] Down - wall[3])

Hint:

1. Left_tile_x, _ = to_tile(x - look_ahead,y) returns tile x of the tile with Red dot. Red dot has same tile_y as pacman
2. to_tile(x, y + look_ahead) returns tile y of the tile below (Blue dot).
3. board[left_tile_x, this_tile_y] returns contents of tile to the left
4. board[left_tile_x, this_tile_y] > 2 returns true if Left tile contains 3,4,5,6,7,8,9,...



```
# 0 = empty black rectangle, 1 = dot, 2 = big dot, 3 = vertical line,
# 4 = horizontal line, 5 = top right, 6 = top left, 7 = bot left, 8 = bottom right
# 9 = ghost house gate
```

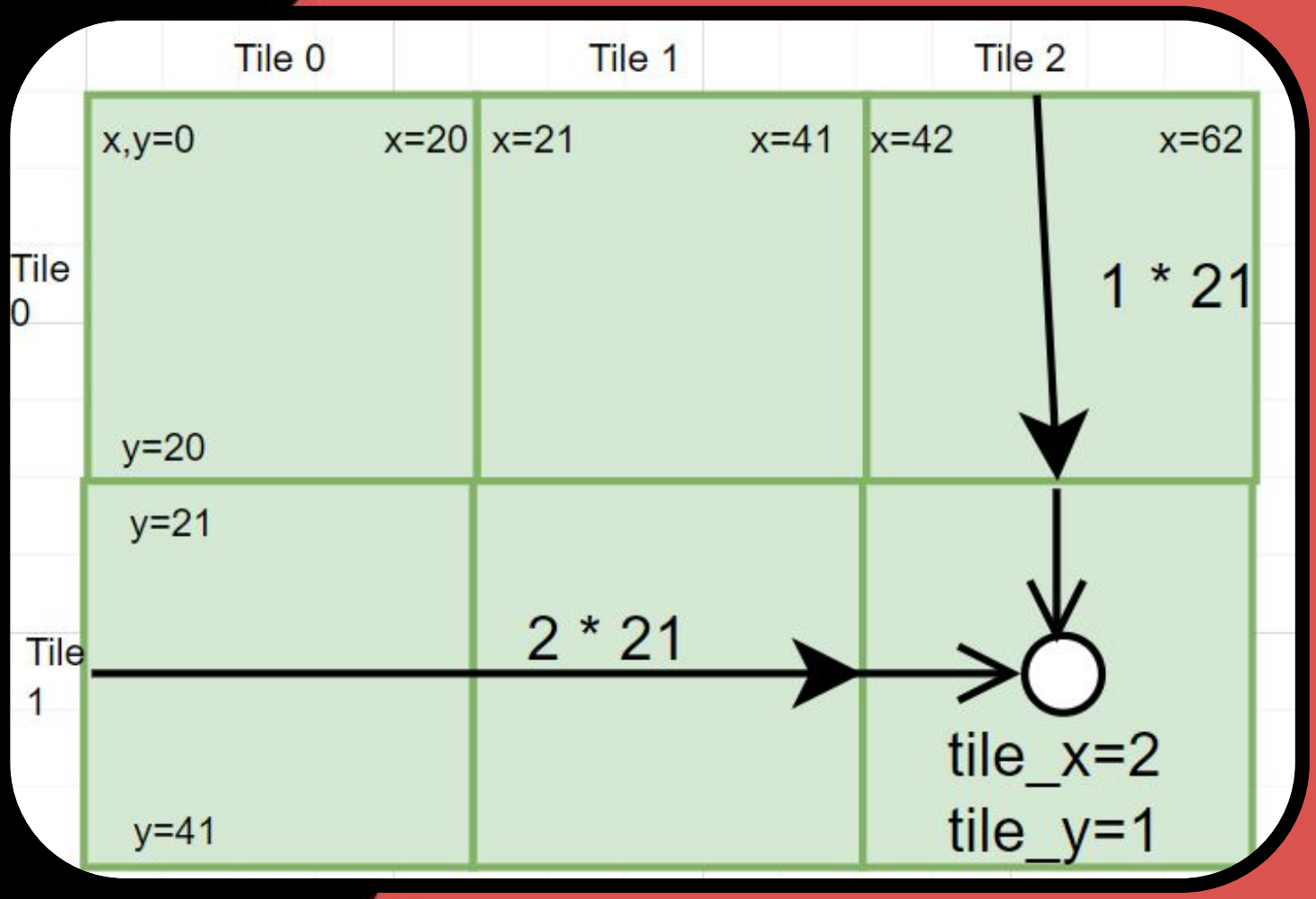
Player class > Move

1. GET keys pressed
2. GET walls
3. IF current direction has wall in same direction
    a. GET this_tile_x, this_tile_y pacman is in.
    b. // Set pacman's center to centerpoint of this tile
    c. SET centerx of pacman to x pixel coordinate of the tile. = TILE_WIDTH * this_tile_x + (width of a tile // 2). Integer division is //
    d. SET centery of pacman to y pixel coordinate of tile pacman is in.
4. SET horizontal = True if moving Left or Right
5. SET vertical = True if moving Up or Down
6. SET Lower= 7 upper = 13

Hint

If pacman is in the tile with white dot.

The centerx of the tile is found by multiplying tile number (tile 2) by width of each tile then add half.



Hint

pygame.key.get_pressed() returns dictionary of keycode to (True/False) values

Get walls with wall_check(pass in centerx and centery of pacman, look_ahead=15 and pacman cannot enter the gate)

Horizontal = (self.direction == ???) or (self.direction == ???)

Move

1. GET keys pressed
2. GET walls
3. IF direction currently facing has a wall
    a. GET TILE (this_tile_x, this_tile_y) pacman is in
    b. SET centerx of pacman to pixel coordinate of this_tile_x PLUS HALF the width of a tile
    c. SET century of pacman to pixel coordinate of this_tile_y PLUS HALF the height of a tile
4. SET horizontal = True if moving Left or Right
5. SET vertical = True if moving Up or Down
6. SET Lower= 7 upper = 13
7. **IF horizontal // Set direction (later slide)**
8. **IF vertical // Set direction (later slide)**
9. // MOVE rect in direction IF no wall in that direction (later slide)
10. IF outside board

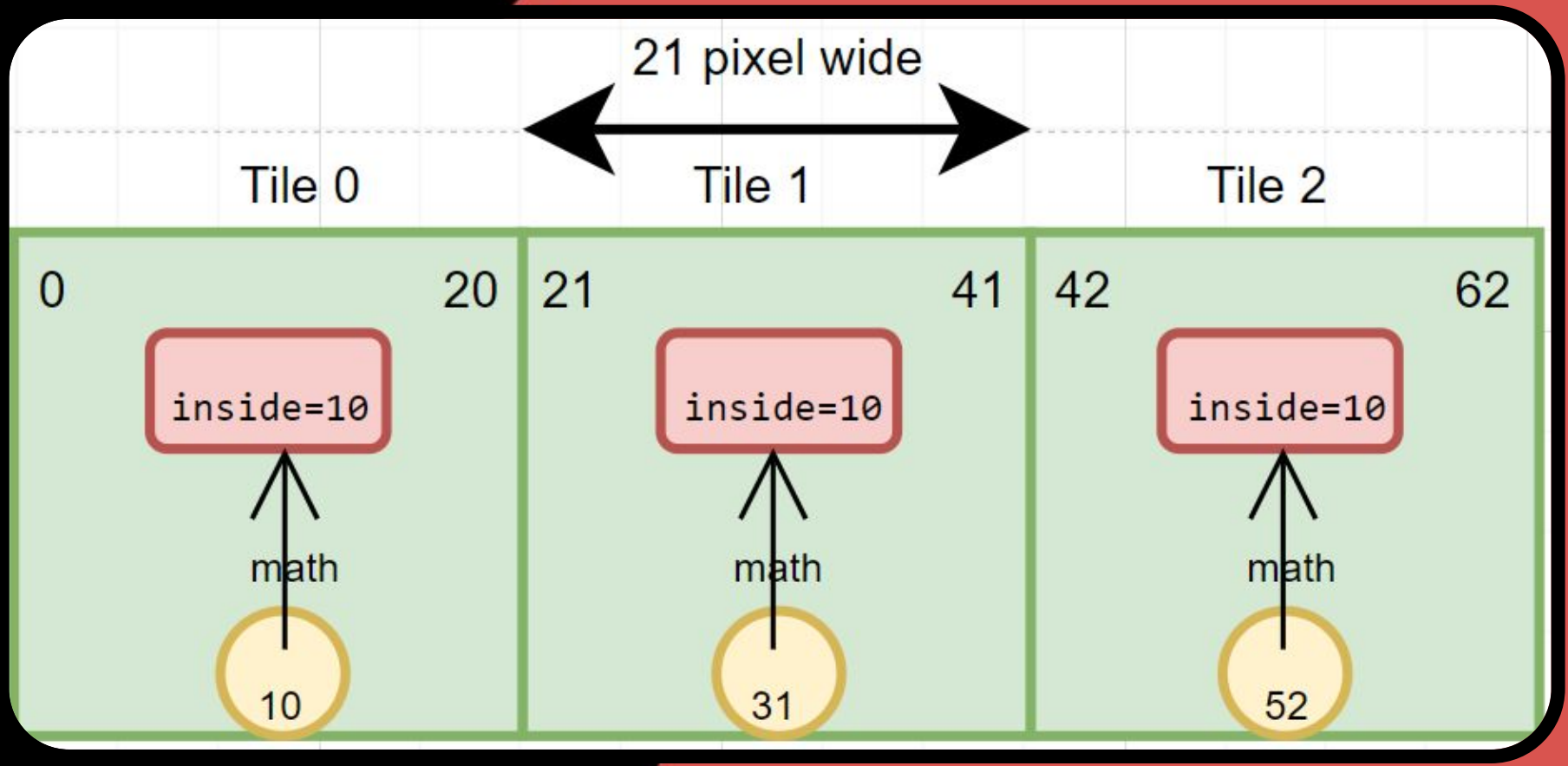
IF HORIZONTAL - set direction

1. (IF HORIZONTAL)
2. IF PRESS A and NO WALL LEFT //Player requests to go left and no wall left then lets go left
    a. SET direction to LEFT
3. IF PRESS D and NO WALL RIGHT
    a. SET direction TO (...)
4. // When turning at a junction, you must be roughly in the center of a tile to turn.
5. IF lower <= SOME_MATH(centerx, TILE_WIDTH) <= upper
    a. IF PRESS W and NO WALL UP
        i. SET direction to UP
    b. IF PRESS S and NO WALL (...)
        i. SET direction TO (...)

Hint (do not proceed to answer on next slide)

The math

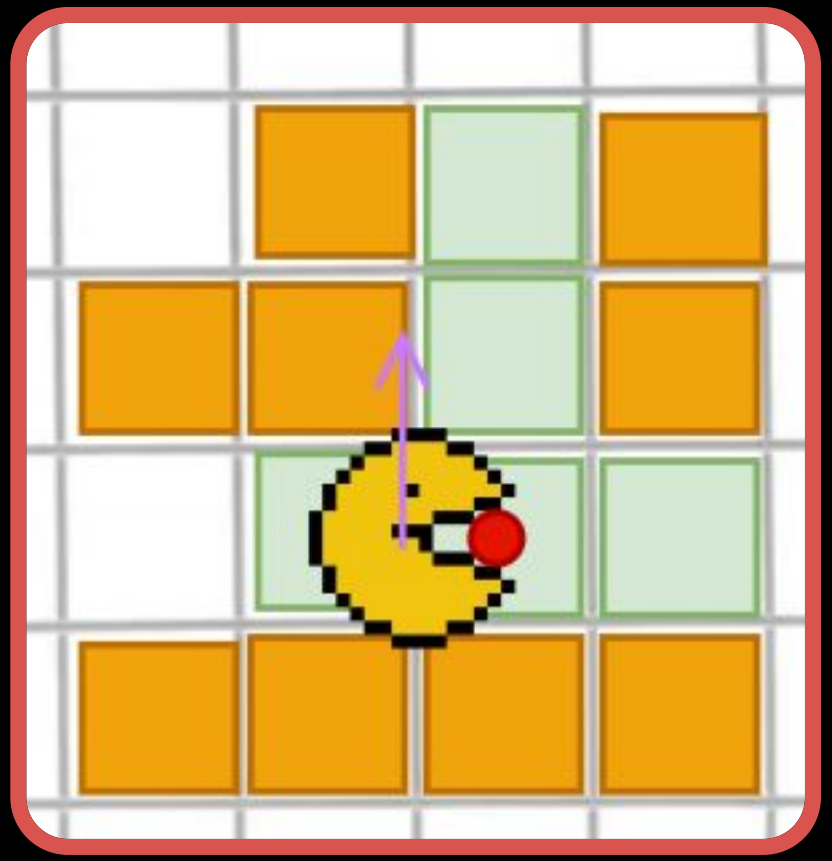
If pacman's position inside a side must be near the center of 10. So between 7 and 13. How do we convert pacman's position (yellow) to the "inside" position



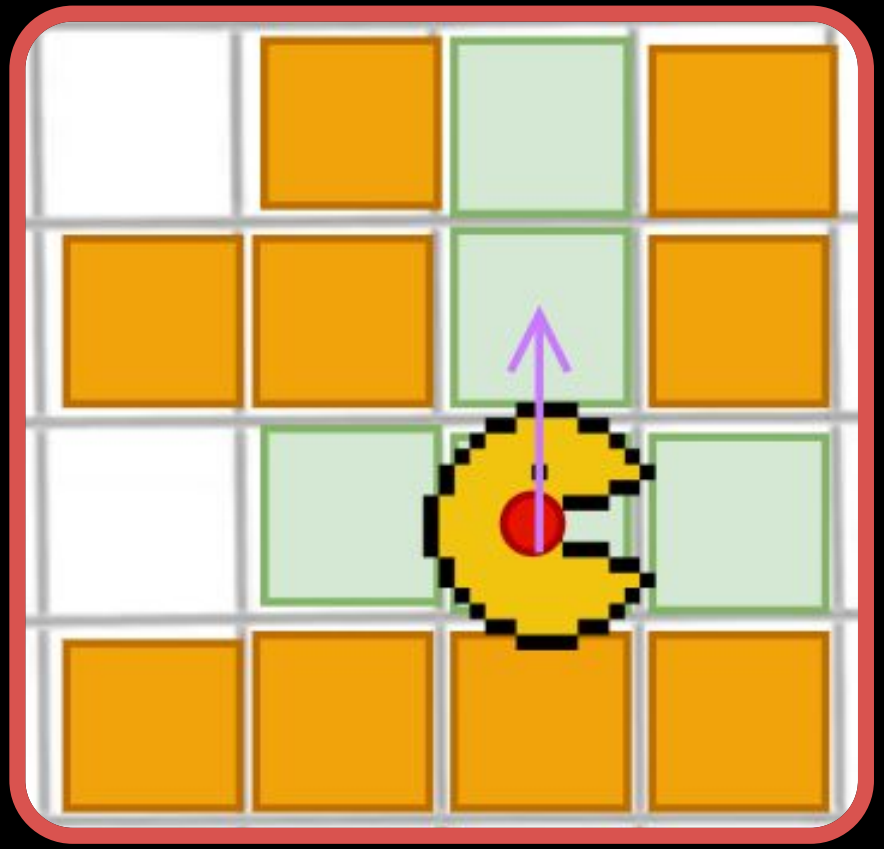
Answer to Math

```
if lower_bound <= (self.rect.centerx % TILE_WIDTH) <=
upper_bound:
```

Junction reasoning



Pacman should not go up until he is in the center of a junction



Pacman is at the center of a junction so can go up

IF VERTICAL - set direction

1. (IF VERTICAL)
2. // When turning at a junction, you must be roughly in the center of a tile to turn.
3. IF lower <= SOME_MATH(centery, TILE_HEIGHT) <= upper
    a. IF PRESS A and NO WALL LEFT
        i. SET direction to LEFT
    b. IF PRESS D and NO WALL RIGHT
        i. SET direction TO RIGHT
4. IF PRESS W and NO WALL UP
    a. SET direction to UP
5. IF PRESS S and NO WALL DOWN
    a. SET direction TO DOWN

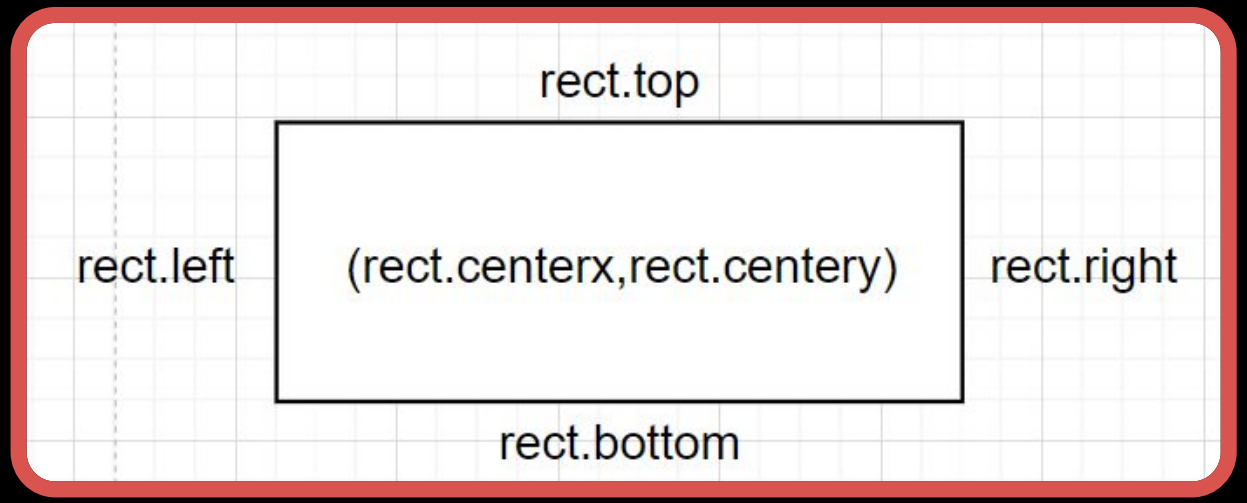
Move

1. GET keys pressed
2. GET walls
3. IF direction currently facing has a wall
    a. GET TILE (this_tile_x, this_tile_y) pacman is in
    b. SET centerx of pacman to pixel coordinate of this_tile_x PLUS HALF the width of a tile
    c. SET century of pacman to pixel coordinate of this_tile_y PLUS HALF the height of a tile
4. SET horizontal = True if moving Left or Right
5. SET vertical = True if moving Up or Down
6. SET Lower= 7 upper = 13
7. IF horizontal // Set direction (later slide)
8. IF vertical // Set direction (later slide)
9. **// MOVE rect in direction IF no wall in that direction (recall last week)**
10. **IF outside board**

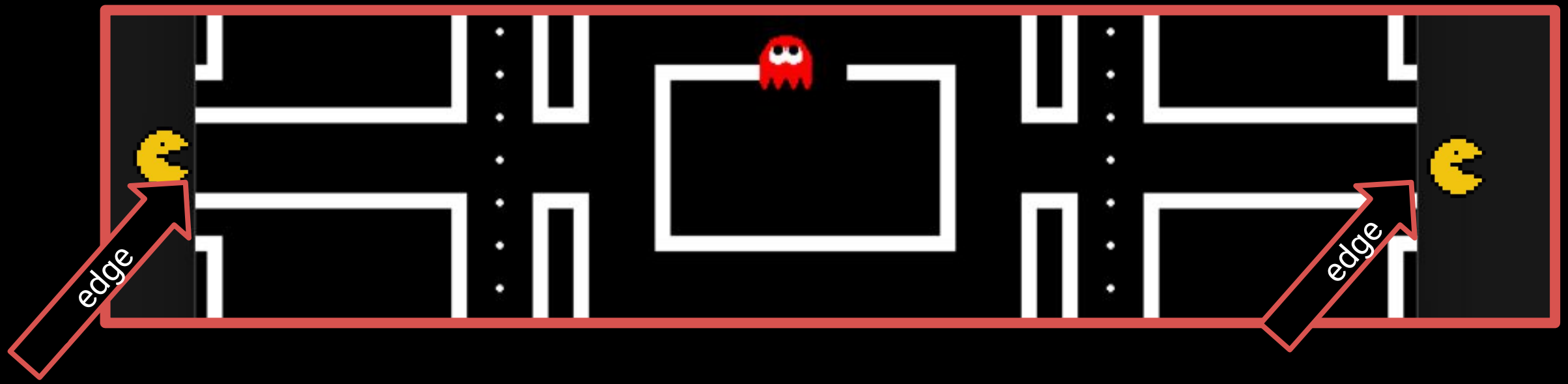
Move and loop around the board via portal

```
// based on last week
IF direction is LEFT and no wall LEFT
        REDUCE centerx by speed
IF direction is RIGHT and no wall RIGHT
        INCREASE centerx by speed
(Write UP and DOWN for centery)
// looping around the board via portal
IF RIGHT EDGE of pacman < 0
        SET RIGHT EDGE of pacman to width of maze
IF LEFT EDGE of pacman > width of maze
        SET RIGHT EDGE of pacman to 0
```

Hint:



Pygame rect (explanation behind self.rect)



<InfPALS/>

Big Project

Session 3

Setting up the ghost (copy the pacman.py)

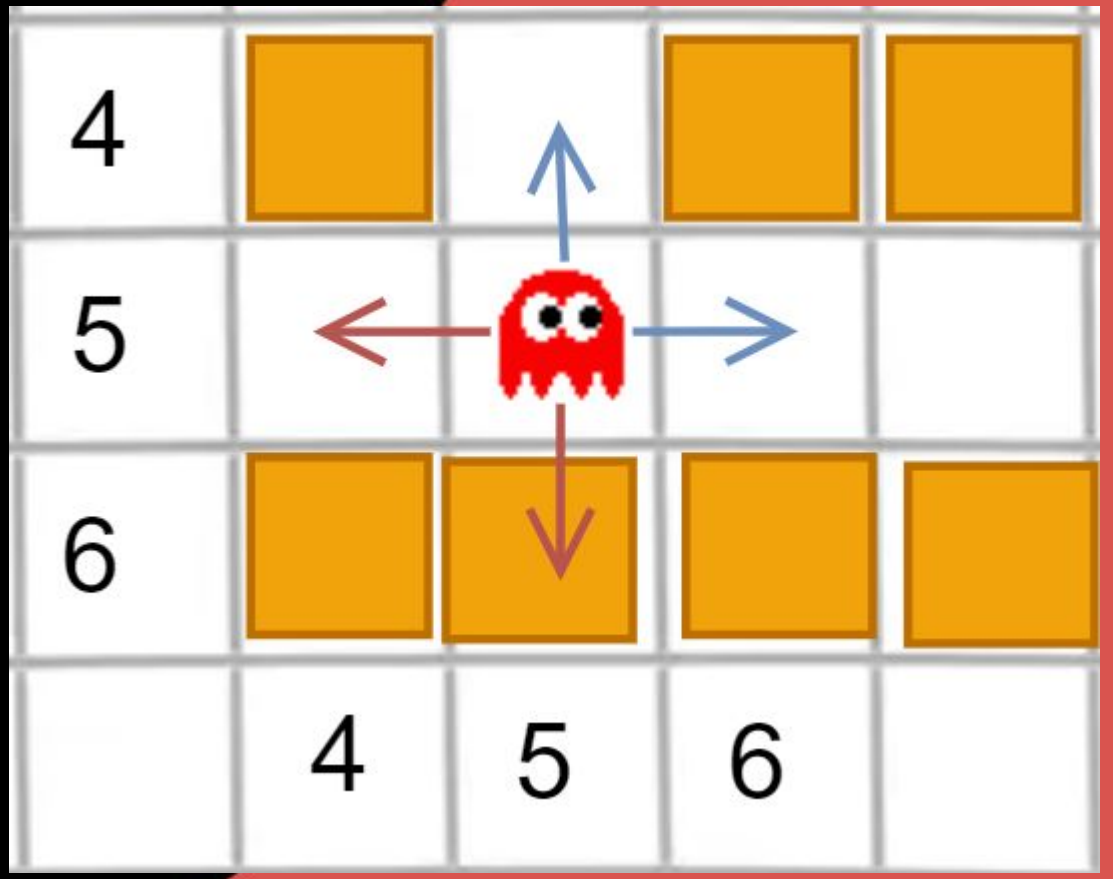
Similar to the player

1. Set up path to folder containing the sprites. There are sprites for looking up/down/left/right, frightened and dead
2. ghost0.png to ghost3.png represent Ghost is alive and looking Left, Right, Up and Down
3. ghost4.png is frightened
4. ghost6.png to ghost9 represents Ghost is dead and looking Left, Right, Up and Down.
5. We store the images in list self.images then use the list index to access the desired image.
6. self.target_direction is the direction the Ghost wants to move (similar to pressing a key on the keyboard) in but it will only move in that direction when there is no wall blocking it.
7. self.house_tile is a tile in the ghost house the Ghost spawns in and returns to regenerate when dead
8. self.leave_house_tile is a tile just outside the Ghost house and used to exit the house
9. Self.mode includes “in_house”, “dead”, “leave_house”, “frightened” and “chas
10. Self.stay_in_house_length = 3 means the Ghost after respawned or at the start of the game stays in the house for 3 seconds before leaving.

get_valid_directions(self, this_tile_x, this_tile_y)

Returns two lists

1. List of valid tiles a ghost may access.
2. List of directions (0 = Left, 1 = Right, 2 = Up, 3 = Down) to access these tiles.
3. The Ghost is at (5,5)
4. Blue arrows are valid tiles and red arrows are invalid.
5. A valid tile is a non-wall tile and not the previous tile
6. The function returns
7. [(5,4), (6,5)] and [2, 1]
8. Meaning we are allowed to go (5,4) using direction 2
9. (Up) and we can go (6,5) using direction 1 (Right)



get_valid_directions(self, this_tile_x, this_tile_y)

1. IF within width of board
    a. SET look_directions = [[x1, y1], [x2, y2], [x3, y3], [x4, y4]] corresponding to one tile Left, Right Up and Down
    b. SET valid_directions and tile_list to empty list
    c. FOR each of the 4 pairs of directions.
        i. Take current position and STEP in X direction
        ii. Take current position and STEP in Y direction
        iii. IF tile at that STEPX and STEPY not wall
            1. IF tile NOT previous tile
                a. ADD direction (0,1,2 or 3) to valid_directions
                b. ADD (STEPX, STEPY) to tile_list
    d. Return tile_list, valid_directions
2. ELSE
3. RETURN 2 empty lists

hint

1. look_directions = [[-1,0], [1,0], [0, -1], [0, 1]] corresponding to Left, Right Up and Down
2. Consider for i in range(4). Notice i goes from 0,1,2,3 (Left, Right, Up, Down).
3. A non wall tile is board[y][x] = 3,4,5,6,7,8
4. Self.previous_tile = (x,y) has the previous_tile

set_frightened_direction(self, this_tile_x, this_tile_y)

Ghosts move to random valid direction at junctions when frightened

1. GET valid directions (list of 0,1,2,3) from get_valid_directions. Ignore the other list
    a. IF valid_directions not empty
    b. Set target_direction to a random direction in the list

Note: handling walls of maze is handled later

hint

Import random

random.choice(mylist)

in_house(self,x,y)

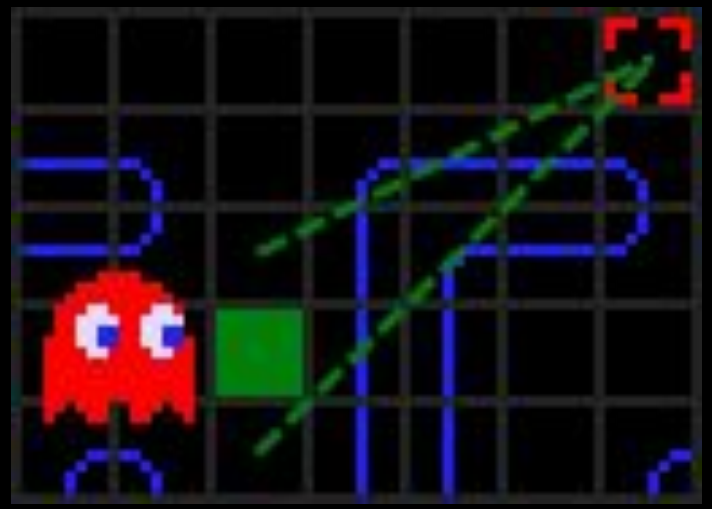
```
def in_house(self, x, y):
    tile_x, tile_y = to_tile(x, y)
    return 12 <= tile_x <= 17 and 14 <= tile_y <= 16
```

Returns True when Ghost is within the boundaries of the house.

go_target_tile(self, this_tile_x, this_tile_y)

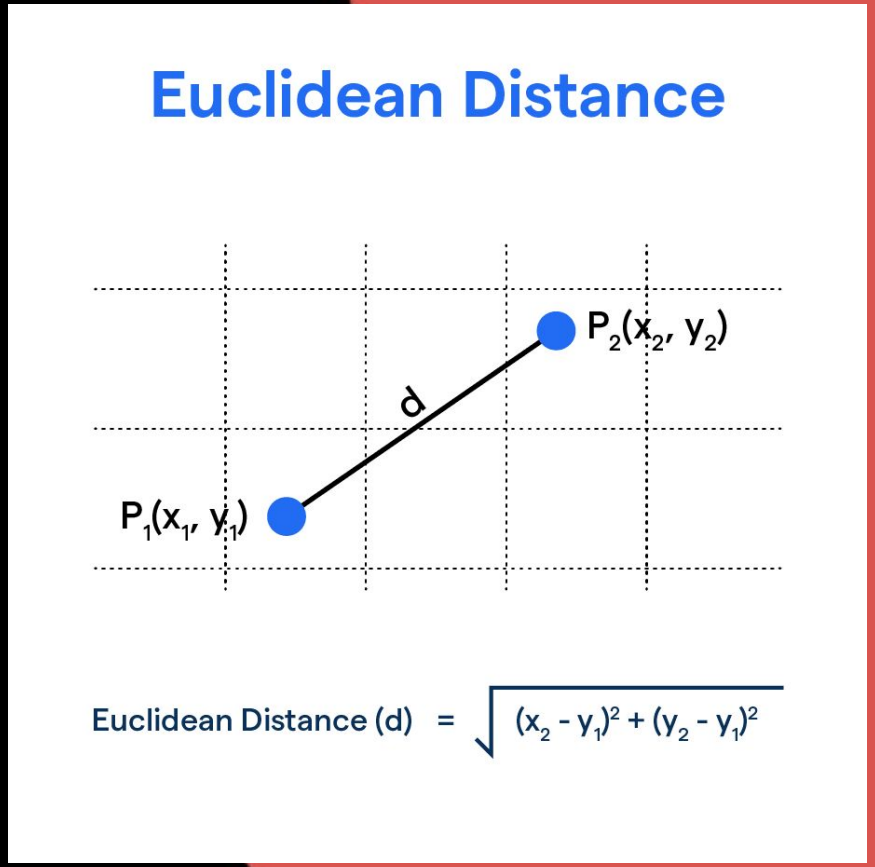
Set self.target_direction to point to tile closest to target tile.

1. GET tile_options and valid_directions from self.get_valid_directions.
2. SET best distance to infinity
3. FOR each tile in tile_options
    a. Calculate euclidean distance from the tile to target_tile
    b. If less than best distance
        i. Update best distance
        ii. Update target_direction to direction (0,1,2,3) corresponding to the tile.



hint

1. float('inf') is infinity
2. Could there be a way to make Euclidean Distance more efficient?
3. For i in range(len(valid_directions)) then notice valid_direction[i] corresponds to coordinates in tile_options[i]
4. Tile_options[i][0] obtains X and tile_options[i][1] obtains Y



Move (copy from pacman.py)

1. Virtually the same as player but
2. If self.mode == “dead” or self.mode == “leave_house” means if the ghost is dead, it can pass through the ghost house gate to enter the ghost house and If the ghost is trying to leave the house it can pass the ghost house gate.
3. In all other cases, it cannot pass the ghost house gate.
4. Notice self.target_direction replaces the keyboard inputs for the player.

Reverse direction (copy from pacman.py)

Reverses direction of ghost. Traditionally occurs when entering frightened mode.

```
    def reverse_direction(self):
        reverse = [1, 0, 3, 2]
        self.direction = reverse[self.direction]
```

frighten(self)

Called by pacman when he eats a power peller

1. IF not dead
    a. SET mode to frightened
    b. CALL reverse_direction

unfrighten(self)

Called when pacman’s powered up timer expires.

1. IF frightened and not in house
    a. SET mode to chase
2. ELSE IF mode is in_house
    a. SET mode to leave house
    b. SET target tile to leave_house_tile

update(self, player: Player, time_delta)

1. GET pacman's x and y tile.
2. GET ghost's current x and y tile
3. IF ghost enters a tile that is not the previous tile
    a. IF mode is chase
        i. SET target_tile to pacman's tile
    b. ELSE IF frightened
        i. CALL set_frightened_direction
    c. ELSE IF dead
        i. IF reached target tile in house
            1. SET mode to in_house
            2. SET house_timer to stay in house for appropriate length

Update part 2

1. ELSE IF in_house
    a. IF house timer has run out
        i. SET mode to leave_house
        ii. SET target_tile to leave house tile
2. ELSE IF leave_house mode
    a. IF reached leave_house_tile
        i. SET mode to chase
3. IF NOT frightened
    a. CALL go_target_tile

SET previous_tile to this tile

IF house_timer above 0 THEN decrement by time delta

IF in same tile as pacman THEN CALL process_collision

CALL self.move()

hint

Get player pixel coordinates with player.rect.centerx, player.rect.centery

Get own pixel coordinates with self.rect.centerx and self.rect.centery

Use to_tile(x,y) to convert pixel coordinates to a tile.

Get mode with self.mode e.g. self.mode == “chase”

If the ghost and pacman are on the same tile, then they have the SAME tile x AND SAME tile y values.

Move with self.move()

process_collision(self, player: Player)

1. IF not dead
    a. IF player powered up
        i. SET my mode to dead
        ii. SET target tile to house_tile
        iii. CALL eat_ghost IN player
    b. ELSE
        i. CALL die IN player

draw(self, window: pygame.Surface) copy from pacman.py

Main difference to player draw is

1. We select an image from self.images based on the state of the ghost (frightened, dead, alive) and the direction the ghost is heading (so that the eyes are facing the right direction)
2. There is a small debug screen at the bottom of the maze.

In Player

1. IF not dead
    a. IF player powered up
        i. SET my mode to dead
        ii. SET target tile to house_tile
        iii. CALL eat_ghost IN player
    b. ELSE
        i. CALL die IN player

Modifications to the player

1. When initialising the player add self.ghost_list = None
2. In self.actions(self, time_delta)
    a. When eating a power pellet
        i. FOR each ghost in ghost_list
            1. CALL ghost.frighten
    b. When powerup ended
        i. FOR each ghost in ghost_list
            1. CALL unfrighten()

Other methods

```
3.      def die(self):
4.          self.alive = False
5.
6.      def eat_ghost(self):
7.          self.score += 200
```

Modifications to the player

1. When initialising the player add self.ghost_list = None
2. In self.actions(self, time_delta)
    a. When eating a power pellet
        i. FOR each ghost in ghost_list
            1. CALL ghost.frighten
    b. When powerup ended
        i. FOR each ghost in ghost_list
            1. CALL unfrighten()

Other methods

```
3.      def die(self):
4.          self.alive = False
5.
6.      def eat_ghost(self):
7.          self.score += 200
```

In main(). Copy from pacman.py

```
    while True:
        # Setting up the game
        # Set up player and ghosts
        player = Player()
        blinky = Ghost()
        ghost_list = [blinky]
        player.ghost_list = ghost_list
```

1. We create player and ghost objects and set up the ghost list

```
# Moving pieces
            player.move()
            player.actions(time_delta)
            for ghost in ghost_list:
                ghost.update(player, time_delta=time_delta)
            # Win and loose condition
            if not player.alive:
                game_is_running = False
            # Check if ate all power pellets
            elif player.pellets_eaten == NUM_PELLETS:
                win = True
                game_is_running = False
```

1. We call update method each ghost in the ghost list.
2. Player wins if all pellets eaten. Player loses if player is not alive.

In main(). Copy from pacman.py

```
            # Render player and ghosts
            player.draw(screen)
            for ghost in ghost_list:
                ghost.draw(screen)
```

1. We call draw method of each ghost

Others

1. Menu and gameover_screen function handle the main menu and game over screen respectively.
2. Sources used when creating this tutorial series.
3. https://www.youtube.com/watch?v=9H27CimgPsQ&t=5171s
4. https://gameinternals.com/understanding-pac-man-ghost-behavior
5. https://pacmancode.com/ (potentially better approach using nodes in a graph instead of looking at walls)