



Cats N' Rats: A Pac-man Inspired Video Game

Authors: Emaan Jivani & Hebat Elkacemi

Northeastern University

EECE 2140: Computing Fundamentals for Engineers

Fall 2024

Date: 11/20/2024

Abstract:

The premise of Cats N' Rats began with the goal to create an already existing game that challenged our graphic, implementation, and programming skills. We wanted to create a game similar to Pac-man to learn gaming language and programming through user interaction, gaming methodologies, and successful teamwork to split tasks up. However, we wanted to reimagine it with a fresh twist and come up with a theme that tells a story that engages players while still keeping the nostalgia associated with the original Pac-man. We used python to program, pygame, a GUI used to interact and create the 2D game, planned a timeline for the project, and implemented data structures learned in class and outside sources like Stackoverflow, W3Schools, and Youtube to fill knowledge gaps. A significant finding we found while working on this project was that we implemented logic based AI rather than modern day Machine Learning AI. This allowed us to appreciate traditional approaches used in classic games. Ultimately, Cats N' Rats reinforced our collaborative, technical, and problem-solving skills.

Contents:

1. [Introduction](#)

[1.1 Background](#)

[1.2 Problem Statement](#)

[1.3 Objectives](#)

[1.4 Scope](#)

2. [Technical Approaches and Code UML](#)

[2.1 Development Environment](#)

[2.2 Data Collection and Preparation](#)

[2.3 Implementation Details](#)

3. [Project Demonstration](#)

[3.1 Screenshots and Code Snippets](#)

4. [Discussion and Future Work](#)

[4.1 Discussion](#)

[4.2 Future Work](#)

5. [Conclusion](#)

6. [Appendix](#)

Chapter 1

Introduction

1.1 Background

Pac-man is a well-known video game originating in Japan, made in 1980 by Tōru Iwatani. This game features a maze-like background with ghosts chasing a small circle that “eats” pellets on the screen which increases the player's score. This game was pivotal to the gaming industry as it was the first of its kind to refocus video games from “shooter” style. This revolution starting with Pac-man created a domino effect for other video games to step away from that style of gaming to try something new and different for that period. Inspired by the appeal of Pac-Man, we created Cats N’ Rats, a game that reimagines Pac-Man with a different story while keeping the same premise as Pac-Man.

1.2 Problem Statement

The purpose of this project was to create a fun and engaging project using Python and the skills acquired in the Computing Fundamentals course. This was done by using the game PacMan as inspiration, where we aimed to challenge ourselves by reimagining the classic arcade game with a different theme. This project allowed us to push the boundaries of our technical abilities as we weren’t very familiar with creating games and working with art. Our focus was to create a fully functional game that combined custom animations, power-ups, and a scoring system. These features added some depth to the gameplay and also helped make the gaming experience more interactive and enjoyable. The project combined creative ideas with strong technical skills to help showcase our ability to combine design, programming, and gameplay into a finished game. This required careful planning, problem-solving, and teamwork, which helped us improve our skills while creating a game that was enjoyable.

1.3 Objectives

The objectives created for this project were to replace PacMan with a rat character and cats, which introduced a unique twist to the classic game. Custom sprites were designed and animated to include characters in different states, like when they were powered up or defeated, which added visual variety and enhanced the gameplay experience. Power-ups, like apples, were also implemented to change the gameplay dynamics by temporarily trapping the cats, which helped to give the rat an advantage. A fully functional game was developed where the rat

navigates through the maze to collect all the cheese pellets while earning points and avoiding the cats to stay alive. A scoring system was also created to track the player's progress, which provided continuous feedback to the player as they continued to navigate through the maze and clear out the maze.

1.4 Scope

This project consisted of designing, developing, and testing a fully functional game inspired by the classic PacMan, using a maze-based layout as a foundation. The code was the most challenging part of the project with it having the largest scope. However, we split up the code into sections using the pseudo-code to reference what we need to accomplish. We used resources at our disposal such as YouTube and Stack Overflow to understand how to have the characters communicate via collisions. These resources helped us troubleshoot issues, fix our code, and gain a better understanding of the best practices for game development. In addition to the coding, the project also consisted of designing custom sprites and animations, a scoring system, and gameplay mechanics like power-ups. The maze layout was also modified to incorporate cheese pellets and apple power-ups to fit the theme and maintain a good balance of challenge and fun. Although the scope of this project consisted of only single-player use, future features like multiplayer modes, extra levels, and a more advanced AI for the cats could be implemented into the game to enhance player experience, however, these features were outside of the scope of our project due to the given timeframe. Using the time frame given, we were still able to create a high-quality cohesive game with a strong foundation to improve in the future.

Chapter 2

Technical Approaches and Code UML

2.1 Development Environment

The development of the game Rats N' Cats used a variety of tools and platforms to code, collaborate, and create different features. The game was built using Python, which was a beginner-friendly programming language, with the PyGame library being a main part of our game to create graphics and bring the game to life. PyGame allowed us to create important game mechanics like character movement, and collision detection, to allow for interactive gameplay and a smooth and functional player experience. The visual components of this game were designed using software called ProCreate as well as PhotoPea to both draw images as well as make them PNGs to use as stickers for game use. Multiple images were drawn representing each of the different states available for the characters as well as the cheese pellets and apple power-ups. To ensure steady progress, the project was split up into phases, which consisted of planning, sprite designing, coding, testing, and debugging. Testing was done regularly as different elements were added to ensure the game worked as intended and that visuals were displayed properly. This structure allowed us to manage the project effectively and create a fully functional final product.

2.2 Data Collection and Preparation

The data collection and preparation primarily consisted of gathering and creating the necessary properties for the game like the sprites and resources to create the borders. Due to the unique theme assigned, the game properties were mainly custom-designed rather than sourced externally. The characters were designed as previously mentioned using ProCreate and PhotoPea, which allowed creative freedom to better fit our theme. Each of the sprites went over multiple iterations to allow for clear, smooth animations, while also being visually appealing and contrasting from the background. Preparation also included creating the maze layout, which was obtained using resources from a similar PacMan project, however, was modified to better fit the theme of the game and to fit the cheese pellets and powerups being added to the maze. As soon as all of the necessary files for preparation were completed, they were ready for use and were easily incorporated into the game.

2.3 Implementation Details

The implementation of the game focused on bringing the conceptual design of Rats N' Cats to life using gameplay mechanics similar to that of PacMan and also visually appealing sprites. This was done using Python as well as PyGame to create an interactive experience between the characters, objects, and the maze. The rat featured multiple animations simulating chomping (closed, half-open, and fully open mouth), while the cat had varying states, which was normal, spooked (when powered down), and defeated (when killed by rat). Cheese pellets and apple power ups were also designed to add visual and gameplay variety. In addition to providing visual feedback to the player constantly throughout the game, these animations were integrated into our game to better fit the theme. The development process also prioritized the key mechanics like movement control, collision detection, and interactions between items. The rat navigated the maze using keyboard inputs, while collision detection was activated when a clear circle around the rat touched the maze borders, cheese pellets, or apple power-ups. These interactions were included as it allowed for there to be a balance between difficulty and reward for a fun user experience. The maze consisted of a grid-based layout, with items placed intentionally depending on their difficulty to allow for the player to use strategic gameplay.

Chapter 3

Project Demonstration

3.1 Screenshots and Code Snippets

In terms of the code of this project, it was a big challenge understanding how we could take our idea and turn it into a reality. However, with research, resources, game programming theories, Python, its built-in libraries, and a GUI, pygame, that takes python code and runs a 2D game, Cats N' Rats came to life.

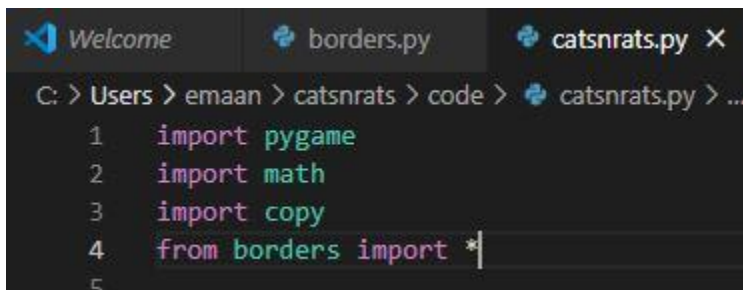
In Figure 1, the output in the terminal when the code is run is shown.

Figure 1:

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active. Below the tabs, there is a prompt 'PS C:\Users\emaan>' followed by the command '& C:/Users/emaan/anaconda3/python.exe c:/Users/emaan/catsnrats/code/catsnrats.py'. The output shows 'pygame 2.6.1 (SDL 2.28.4, Python 3.12.4)' and 'Hello from the pygame community. https://www.pygame.org/contribute.html'.

We also imported various libraries through python and pygame exemplified below in Figure 2.

Figure 2:

A screenshot of a code editor with a dark background. The top bar shows three tabs: 'Welcome', 'borders.py', and 'catsnrats.py'. The 'catsnrats.py' tab is active. The code in the editor shows the following imports: 'import pygame', 'import math', 'import copy', and 'from borders import *'. The cursor is at the end of the fourth line.

In terms of the code itself, we initialized a class called `_cat_` which encompassed the main functions the cat does such as dealing with collisions, targeting the rat, drawing, and moving the

various cats. This is shown in the [Appendix](#) in [Figures 3-9](#). Nearly half the entire game's code was used to initialize the various cat states and how it moves at the control of the player's keys. It was a very crucial part to get correct. For clarity and conciseness, we have shown only the first block of code from each definition in the `_cat_` class as shown below in the Appendix.

We used an abundant amount of if statements to complete the programming which meant very long code blocks and functions. This method was not the most efficient. We could have used more loop based programming with for statements or nested while loops. Nonetheless, the code still did run.

Moving towards the miscellaneous functions to the game, a function `_draw_misc_` was created to deal with scoring, and the end of game screens shown below in Figure 10.

Figure 10:

```
628 # this function draws miscellaneous things like the scoring text, images of rat as lives, lives
629 # left and game over and victory screens
630 def draw_misc():
631     # displays score
632     score_text = font.render(f'Score: {score}', True, '#00e6fc')
633     screen.blit(score_text, (10, 920))
634
635     # displays power up indication
636     if powerup:
637         screen.blit(superapple_img, (140, 910))
638
639     # displays number of lives left
640     for i in range(lives):
641         screen.blit(pygame.transform.scale(
642             rat_images[0], (40, 40)), (650 + i * 40, 915))
643     lives_text = font.render(f'{lives} lives left', True, '#FFFFFF') # Number of lives
644     screen.blit(lives_text, (650 + lives * 40 + 10, 920)) # Adjust position to align with icons
645
646     # displays game over screen
647     if game_over:
648         pygame.draw.rect(screen, "#d52b1e",
649                             [225, 390, 450, 90], 0, 10)
650         gameover_text = font.render(
651             'Game Over! Click the Space Bar to Restart', True, '#000000')
652         screen.blit(gameover_text, (240, 430))
653
654     # displays victory screen
655     if victory:
656         pygame.draw.rect(screen, "#339c21",
657                             [225, 390, 450, 90], 0, 10)
658         victory_text = font.render(
659             'Victory! Click the Space Bar to Restart', True, '#000000')
660         screen.blit(victory_text, (240, 430))
```

The next block of code introduces and initializes the border or maze you see in the game. This was done using resources at our disposal while learning the matrix components and

understanding where the math python library comes in. In Figure 11, the borders are separated into 9 sections as shown below.

Figure 11:

```
680 # method that draws the borders (maze)
681 def draw_borders():
682     num1 = ((HEIGHT - 50) // 32)
683     num2 = (WIDTH // 30)
684     for i in range(len(level)):
685         for j in range(len(level[i])):
686
687             # draws cheese pellets
688             if level[i][j] == 1:
689                 screen.blit(cheese_img, (j * num2 + (0.5 * num2) - 10, i * num1 + (0.5 * num1) - 10))
690             # draws apple pellets
691             if level[i][j] == 2 and not flicker:
692                 screen.blit(superapple_img, (j * num2 + (0.5 * num2) - 10, i * num1 + (0.5 * num1) - 10))
693             if level[i][j] == 3:
694                 pygame.draw.line(screen, color,
695                                 (j * num2 + (0.5 * num2),
696                                 i * num1),
697                                 (j * num2 + (0.5 * num2),
698                                 i * num1 + num1), 3)
699             if level[i][j] == 4:
700                 pygame.draw.line(screen, color,
701                                 (j * num2, i * num1 + (0.5 * num1)),
702                                 (j * num2 + num2,
703                                 i * num1 + (0.5 * num1)), 3)
704             if level[i][j] == 5:
705                 pygame.draw.arc(screen, color,
706                                 [(j * num2 - (num2 * 0.4)) - 2,
707                                 (i * num1 + (0.5 * num1)),
708                                 num2, num1],
709                                 0, PI / 2, 3)
710             if level[i][j] == 6:
711                 pygame.draw.arc(screen, color,
712                                 [(j * num2 + (num2 * 0.5)),
713                                 (i * num1 + (0.5 * num1)),
```

Next, the rat was coded to move according to the output of the device's arrow keys as shown in Figure 12. The function `_move_rat_` allowed for player interaction and immersion into the game for that nostalgic feel.

Figure 12:

```

799 # moves rat based on user clicked keys
800 v def move_rat(play_x, play_y):
801 v     if direction == 0 and turns_allowed[0]:
802         play_x += rat_speed # right
803 v     elif direction == 1 and turns_allowed[1]:
804         play_x -= rat_speed # left
805 v     if direction == 2 and turns_allowed[2]:
806         play_y -= rat_speed # up
807 v     elif direction == 3 and turns_allowed[3]:
808         play_y += rat_speed # down
809     return play_x, play_y
810

```

The next part in our code was the `_get_targets_` method shown in Figure 13. This method allowed the cats to “look for” the rat initiating AI logic into our game. It checks for which state the cat is in, blue, yellow, orange, purple, dead, powerup, etc, and follows the rat (or doesnt) wherever it is.

Figure 13:

```

811 # calculates target positions for each cat based on state of the game and rat position
812 def get_targets(orange_x, orange_y, purple_x, purple_y,
813                blue_x, blue_y, yellow_x, yellow_y):
814     if rat_x < 450:
815         runaway_x = 900
816     else:
817         runaway_x = 0
818     if rat_y < 450:
819         runaway_y = 900
820     else:
821         runaway_y = 0
822     return_target = (380, 400)
823     if powerup:
824         # orange cat targets rat
825         if not orange.dead and not eaten_cat[0]:
826             orange_target = (runaway_x, runaway_y)
827         elif not orange.dead and eaten_cat[0]:
828             if 340 < orange_x < 560 and 340 < orange_y < 500:
829                 orange_target = (400, 100)
830             else:
831                 orange_target = (rat_x, rat_y)
832         else:
833             orange_target = return_target
834
835     # purple cat targets rat
836     if not purple.dead and not eaten_cat[1]:
837         purple_target = (runaway_x, rat_y)
838     elif not purple.dead and eaten_cat[1]:
839         if 340 < purple_x < 560 and 340 < purple_y < 500:
840             purple_target = (400, 100)
841         else:
842             purple_target = (rat_x, rat_y)
843     else:

```

Finally, approaching the end of the program, Figure 14 shows the entire screen being initialized, images are loaded, background is created, font is added, and timer is programmed.

Figure 14:


```

906 # sets position, initializes screen, loads cats/rat images
907 if __name__ == '__main__':
908     pygame.init()
909     WIDTH = 900
910     HEIGHT = 950
911     screen = pygame.display.set_mode([WIDTH, HEIGHT])
912     timer = pygame.time.Clock()
913     fps = 60
914     font = pygame.font.Font('freesansbold.ttf', 20)
915     level = copy.deepcopy(borders)
916     color = '#0000a6'
917     PI = math.pi
918     rat_images = []
919     for i in range(1, 5):
920         rat_images.append(pygame.transform.scale(pygame.image.load(
921             f'catsnrats/img/{i}.png'), (50, 50)))
922         orange_img = pygame.transform.scale(pygame.image.load(
923             f'catsnrats/img/orange.png'), (50, 50))
924         blue_img = pygame.transform.scale(pygame.image.load(
925             f'catsnrats/img/blue.png'), (50, 50))
926         purple_img = pygame.transform.scale(pygame.image.load(
927             f'catsnrats/img/purple.png'), (50, 50))
928         yellow_img = pygame.transform.scale(pygame.image.load(
929             f'catsnrats/img/yellow.png'), (50, 50))
930         powerup_img = pygame.transform.scale(pygame.image.load(
931             f'catsnrats/img/powerup.png'), (50, 50))
932         dead_img = pygame.transform.scale(pygame.image.load(
933             f'catsnrats/img/dead.png'), (50, 50))

```

When the rat eats the big apple powerups, the cats can be temporarily defeated and their speeds change. The cat at its normal state is contingent with the speed of the rat. When the cat is in Powerup state, it is half the speed. Finally, when the cats are defeated, they go back to the box in the center of the maze at double their original speed. This was shown in this code below in Figure 15.

Figure 15:

```

1009         # speed of cat when it is a powerup (slows down)
1010         if powerup:
1011             cat_speeds = [1, 1, 1, 1]
1012         # speed of cat normally (same speed as rat)
1013         else:
1014             cat_speeds = [2, 2, 2, 2]
1015         if eaten_cat[0]:
1016             cat_speeds[0] = 2
1017         if eaten_cat[1]:
1018             cat_speeds[1] = 2
1019         if eaten_cat[2]:
1020             cat_speeds[2] = 2
1021         if eaten_cat[3]:
1022             cat_speeds[3] = 2
1023         if orange_dead:
1024             cat_speeds[0] = 4
1025         if purple_dead:
1026             cat_speeds[1] = 4
1027         if blue_dead:
1028             cat_speeds[2] = 4
1029         if yellow_dead:
1030             cat_speeds[3] = 4
1031         victory = True

```

To make the rats able to detect collisions between the walls and all variations of the cats, we needed to come up with a way to wrap a border around the rat to do so. We used shapes and transparency to achieve this as shown below in Figure 16.

Figure 16:

```

1037         # draws transparant margin
1038         rat_transparant_margin = pygame.Surface((50,50), pygame.SRCALPHA)
1039         transparent_color = (13, 17, 23, 0)
1040         rat_radius = 20
1041         rat_rect = pygame.Rect(center_x - rat_radius, center_y - rat_radius, rat_radius * 2, rat_radius * 2)
1042         screen.blit(rat_transparant_margin, (center_x - 25, center_y - 25))

```

The end of the code was simply looping the game using all the functions we created as well as initializing positioning of different elements. To create the cheese and apple pellet sprites, those were loaded at the beginning of the program to add to the maze with the correct positioning.

A demo of the game is shown [below](#) in the Appendix.

Chapter 4

Discussion and Future Work

4.1 Discussion

Our project allowed us to blend our technical and creative skills to create a game similar to the class arcade game PacMan. This was done by using custom sprites and animations and modifying various features of the game to better fit the theme for Rats N' Cats. This was done by designing various sprites and objects to incorporate into the game and creating new power-ups like apples and cheese as pellets to align with the theme. Players were also able to interact with the game by navigating through the maze and interacting with the pellets, power-ups, and cats, like being able to disable the cats temporarily or collecting cheese pellets to increase their score. The live scoring system also allows for a better player experience as it gives the player continuous feedback, which as a result, motivates players to improve their performance and continue engaging with the game.

Although our project was a success, we experienced numerous challenges along the way, in specific, debugging and ensuring the game ran smoothly between sprites and the maze. A main feature we encountered challenges with was collision detection as it required a clear ring around the rat to allow us to change the border, without giving away how the rat was colliding with other sprites. This was done using a resource known as StackExchange to help us troubleshoot this issue and better understand how to improve the quality of the collision detection. We also detected a bug in the code we could not distinguish in which the rat would get “stuck” trying to turn and corner around the maze.

4.2 Future Work

We learned a great deal during this project about implementing project methods and creating something from start to finish. In terms of key findings, we found an appreciation for the use of technical and artistic methods to create something to allow for a greater user experience. Using Github in this project allowed us to feel safe when going through many interactions with the code and images. Concerning any future projects, Github has proved to be a very helpful tool for organizing data and moving toward a polished project.

In the future, we want to prioritize adding more elements that add difficulty. For example, adding more cats increases the amount of power-ups for both the rat and the cats. We also thought of creating a level system or a gradual speed increase for all elements garnering a fast-paced, stressful, almost addicting environment.

Chapter 5

Conclusion

In conclusion, the Rats N' Cats project successfully achieved its goal of reimagining the classic Pac-Man game with a creative twist. This was accomplished by incorporating custom-designed characters, animations, and gameplay mechanics, which provided a unique gaming experience that blended technical skills with creativity. By implementing elements such as power-ups and a live scoring system, the project enhanced gameplay and demonstrated a solid understanding of programming concepts and design principles. Although the project faced challenges like debugging, collision detection, and optimizing animations, these obstacles were addressed through iterative development and teamwork, resulting in a polished final product. The game also established a strong foundation for future improvements and upgrades, such as enhanced cat AI behavior, multiplayer modes, and increased difficulty levels to improve player experience. Overall, the project highlighted our ability to manage complex tasks, solve problems, and create a cohesive, fully functional final product, showcasing its success as both a learning experience and a completed project.

Bibliography

“PacMan in Python and Pygame.”, YouTube, 8 Feb. 2024,
youtu.be/KB0BHw7rnqU?si=SFhu3D7VDnoRB-Ju+s.

“Gettingstarted - Wiki.” GettingStarted - Pygame Wiki, www.pygame.org/wiki/GettingStarted.

“Collision Detection with Non-Rectangular Images.” Game Development Stack Exchange, 1 Aug. 1957,
gamedev.stackexchange.com/questions/30866/collision-detection-with-non-rectangular-images.

Appendix

Figure 3:

```
11 class cat:
12     def __init__(self, x_coord, y_coord, target, speed, img, direct, dead, box, id):
13         self.x_pos = x_coord
14         self.y_pos = y_coord
15         self.center_x = self.x_pos + 30
16         self.center_y = self.y_pos + 30
17         self.target = target
18         self.speed = speed
19         self.img = img
20         self.direction = direct
21         self.dead = dead
22         self.in_box = box
23         self.id = id
24         self.turns, self.in_box = self.check_collisions()
25         self.rect = self.draw_cats()
```

Figure 4:

```
27 # method to draw cats as dead, powerup, or normal
28 def draw_cats(self):
29     if (not powerup and not self.dead) or (eaten_cat[self.id] and powerup and not self.dead):
30         screen.blit(self.img, (self.x_pos, self.y_pos))
31     elif powerup and not self.dead and not eaten_cat[self.id]:
32         screen.blit(powerup_img, (self.x_pos, self.y_pos))
33     else:
34         screen.blit(dead_img, (self.x_pos, self.y_pos))
35
36     # variable to create a clear rectangle around cat image to detect collision
37     cat_rect = pygame.rect.Rect(
38         (self.center_x - 10, self.center_y - 10), (20, 20))
39     return cat_rect
```

Figure 5:

```
41 def check_collisions(self):
42     num1 = ((HEIGHT - 50) // 32)
43     num2 = (WIDTH // 30)
44     num3 = 15
45     self.turns = [False, False, False, False] # right, left, up, down
46
47     # self[0] -> right
48     # self[1] -> left
49     # self[2] -> up
50     # self[3] -> down
51     if 0 < self.center_x // 30 < 29:
52         if level[(self.center_y - num3) // num1][self.center_x // num2] == 9:
53             self.turns[2] = True
54         if level[self.center_y // num1][(self.center_x - num3) // num2] < 3 \
55             or (level[self.center_y // num1][(self.center_x - num3) // num2] == 9 and (
56                 self.in_box or self.dead)):
57             self.turns[1] = True
58         if level[self.center_y //
59                 num1][(self.center_x + num3) // num2] < 3 \
60             or (level[self.center_y //
61                     num1][(self.center_x + num3) // num2] == 9 and (
62                 self.in_box or self.dead)):
63             self.turns[0] = True
64         if level[(self.center_y + num3) //
65                 num1][self.center_x // num2] < 3 \
66             or (level[(self.center_y + num3) //
67                     num1][self.center_x // num2] == 9 and (
68                 self.in_box or self.dead)):
69             self.turns[3] = True
70         if level[(self.center_y - num3) //
71                 num1][self.center_x // num2] < 3 \
72             or (level[(self.center_y - num3) //
```

Figure 6:

```
140 # method directs yellow cat, targets rat, and changed direction if collides
141 def move_yellow(self):
142     if self.direction == 0:
143         if self.target[0] > self.x_pos and self.turns[0]:
144             self.x_pos += self.speed
145         elif not self.turns[0]:
146             if self.target[1] > self.y_pos and self.turns[3]:
147                 self.direction = 3
148                 self.y_pos += self.speed
149             elif self.target[1] < self.y_pos and self.turns[2]:
150                 self.direction = 2
151                 self.y_pos -= self.speed
152             elif self.target[0] < self.x_pos and self.turns[1]:
153                 self.direction = 1
154                 self.x_pos -= self.speed
155             elif self.turns[3]:
156                 self.direction = 3
157                 self.y_pos += self.speed
158             elif self.turns[2]:
159                 self.direction = 2
160                 self.y_pos -= self.speed
161             elif self.turns[1]:
162                 self.direction = 1
163                 self.x_pos -= self.speed
164         elif self.turns[0]:
165             if self.target[1] > self.y_pos and self.turns[3]:
166                 self.direction = 3
167                 self.y_pos += self.speed
168             if self.target[1] < self.y_pos and self.turns[2]:
169                 self.direction = 2
170                 self.y_pos -= self.speed
```

Figure 7:

```
278 # method directs orange cat, targets rat, and changed direction if collides
279 def move_orange(self):
280     if self.direction == 0:
281         if self.target[0] > self.x_pos and self.turns[0]:
282             self.x_pos += self.speed
283         elif not self.turns[0]:
284             if self.target[1] > self.y_pos and self.turns[3]:
285                 self.direction = 3
286                 self.y_pos += self.speed
287             elif self.target[1] < self.y_pos and self.turns[2]:
288                 self.direction = 2
289                 self.y_pos -= self.speed
290             elif self.target[0] < self.x_pos and self.turns[1]:
291                 self.direction = 1
292                 self.x_pos -= self.speed
293             elif self.turns[3]:
294                 self.direction = 3
295                 self.y_pos += self.speed
296             elif self.turns[2]:
297                 self.direction = 2
298                 self.y_pos -= self.speed
299             elif self.turns[1]:
300                 self.direction = 1
301                 self.x_pos -= self.speed
302             elif self.turns[0]:
303                 self.x_pos += self.speed
304     elif self.direction == 1:
305         if self.target[0] < self.x_pos and self.turns[1]:
306             self.x_pos -= self.speed
307         elif not self.turns[1]:
308             if self.target[1] > self.y_pos and self.turns[3]:
309                 self.direction = 3
```


Figure 8:

```
383 # method directs purple cat, targets rat, and changed direction if collides
384 def move_purple(self):
385     if self.direction == 0:
386         if self.target[0] > self.x_pos and self.turns[0]:
387             self.x_pos += self.speed
388         elif not self.turns[0]:
389             if self.target[1] > self.y_pos and self.turns[3]:
390                 self.direction = 3
391                 self.y_pos += self.speed
392             elif self.target[1] < self.y_pos and self.turns[2]:
393                 self.direction = 2
394                 self.y_pos -= self.speed
395             elif self.target[0] < self.x_pos and self.turns[1]:
396                 self.direction = 1
397                 self.x_pos -= self.speed
398             elif self.turns[3]:
399                 self.direction = 3
400                 self.y_pos += self.speed
401             elif self.turns[2]:
402                 self.direction = 2
403                 self.y_pos -= self.speed
404             elif self.turns[1]:
405                 self.direction = 1
406                 self.x_pos -= self.speed
407         elif self.turns[0]:
408             if self.target[1] > self.y_pos and self.turns[3]:
409                 self.direction = 3
410                 self.y_pos += self.speed
411             if self.target[1] < self.y_pos and self.turns[2]:
412                 self.direction = 2
413                 self.y_pos -= self.speed
414         else:
415             self.x_pos += self.speed
```

Figure 9:

```
504 # method directs blue cat, targets rat, and changed direction if collides
505 def move_blue(self):
506     if self.direction == 0:
507         if self.target[0] > self.x_pos and self.turns[0]:
508             self.x_pos += self.speed
509         elif not self.turns[0]:
510             if self.target[1] > self.y_pos and self.turns[3]:
511                 self.direction = 3
512                 self.y_pos += self.speed
513             elif self.target[1] < self.y_pos and self.turns[2]:
514                 self.direction = 2
515                 self.y_pos -= self.speed
516             elif self.target[0] < self.x_pos and self.turns[1]:
517                 self.direction = 1
518                 self.x_pos -= self.speed
519             elif self.turns[3]:
520                 self.direction = 3
521                 self.y_pos += self.speed
522             elif self.turns[2]:
523                 self.direction = 2
524                 self.y_pos -= self.speed
525             elif self.turns[1]:
526                 self.direction = 1
527                 self.x_pos -= self.speed
528             elif self.turns[0]:
529                 self.x_pos += self.speed
530         elif self.direction == 1:
531             if self.target[1] > self.y_pos and self.turns[3]:
532                 self.direction = 3
533             elif self.target[0] < self.x_pos and self.turns[1]:
534                 self.x_pos -= self.speed
535             elif not self.turns[1]:
536                 if self.target[1] > self.y_pos and self.turns[3]:
```

Video Demonstration:

<https://youtube.com/shorts/BSvIHtnLZYM?feature=share>