CRITERIA	EXEMPLARY (90-100)	SATISFACTORY (80-89)	DEVELOPING (70-79)	BEGINNING (below 70)	WEIGHT
Experimental Plan (Flowchart/ Algorithm) ***(SO-PI: B1)***	Experimental plan has supporting details and diagram/algorithm that is stated and well explained	Experimental plan has supporting details and diagram/ algorithm that is stated but not explained	Experimental plan is vague or brief. It has supporting details and does not have diagram/ algorithm	No experimental plan presented	20%
Codes/Data/ Program	Data is well utilized in the program. Program code are easy to read. Program output has no error. Questions are answered completely and correctly	Data is somewhat utilized in the program. Program code are easy to read. Program output has an output but logically incorrect. Some questions are answered completely and correctly	Data is not utilized in the program. It has a missing significant code/syntax in the program.	No program presented	30%

Use of Appropriate Tools and Techniques ***(SO-PI: K1)***	Appropriate tools and techniques are properly used for all aspects of the project	Appropriate tools and techniques are used in most of the aspects of the project and all of these are used properly	Appropriate tools and techniques are used in majority of the aspects of the project and all of these are used properly	Appropriate tools and techniques are used in less than half of the aspects of the project and/or tools are not used properly in at least half the aspects of the project.	10%
Project Documentation	Project documentation is orderly presented starting from statement of the problem, to objective of the project, followed by review of literature, design consideration, presentation of data or output and conclusion. The report was grammatically	Project documentation is complete with statement of the problem, objectives, design consideration, presentation of data and output and conclusion. The report had minimal grammatical errors and somewhat presented logically. The required format was used.	Project documentation is basically limited to algorithm presentation of data and output but no basis of the design was presented. The report had a lot of grammatical errors and not logically presented; the required format was barely used.	Project documentation is not reflective of algorithm design and/or characterization. The report had a lot of grammatical errors, was not logically presented and the required format was not used.	30 %

	correct, logically presented and used the required format.				
Project Presentation SO-PI: G2	Project presentation is complete and backed up by complete design consideration, logic formulation and review of related literature.	Project presentation is complete with algorithm simulation results backed up by design considerations.	Project presentation shows a system completely simulated but is not backed up by clear explanation of how algorithm was derived	Project presentation lacks clarity in terms of presenting and characterizing the behavior of the algorithm	10 %

DE LA SALLE UNIVERSITY - MANILA

MULTI PAC-MAN

A Term Project
Presented to Engr. Ramon Stephen Ruiz
In Partial Fulfillment of the
Requirements for the Course Programming Logic and Design Laboratory (LBYCPA1)

---

by
Tabanao, Leigh Andrei M. –
Tiu, Timothy Brian A. –
EQ1
TF: 7:30 AM – 10:30 AM

APRIL 14, 2023

Table of Contents

List of Figures

List of Tables

I. Introduction

The project proposed by the students is a Python recreation of the classic hit game Pac-Man, but with some twists from other classic video games. The students also propose the usage of the Python library "Pygame" alongside the lessons of the course program, to improve the graphics for the game. For a brief description of the original game: the player will be controlling Pac-Man, a yellow ball tasked to collect or eat all the cookies (yellow dots) in a maze as quickly as possible. However, in the game, there are ghosts, colored pink, orange, red, and cyan, roaming the maze. The player shall help Pac-Man evade, as touching these ghosts will result in losing a "life." The game is over when all of Pac-Man's lives are lost, and the score is based on the number of cookies that have been eaten.

A. Background of the Study

The popularity of gaming applications are increasing quickly along with the development of more advanced technologies and gadgets. Different types of games have also emerged virtually and physically, games can now be played in our gadgets such as laptops, tablets, and even mobile phones instead of going to game arcades or game machines. Modern gaming applications teach the brain to concentrate and think quickly in addition to providing amusement. These applications are not something that most people are familiar with (Wulandari & Harnadi, 2014). The gaming industry has truly evolved throughout the years and has attracted many gamers of all age groups, including adults and children.

The goal of the project is to create a twist or unique mechanic of this game recreation which involves the addition of multiple continuous pages and/or screen scrolling, akin to other classic video games such as Super Mario Bros and The Legend of Zelda. This is where the title of the project, "Multi Pac-Man," comes from. Furthermore, the students might redesign the other screens or pages into the theme of Super Mario Bros., or Zelda, adding "multiversal travel" into the theme of the game, further fitting the title "Multi Pac-Man."

B. Problem Statement

Different versions of the classic Pacman can be played on different machines as well as different devices, a modified version of the game will be developed in python language to allow python users to enjoy the game as well as enable them to try out a revamped version of Pacman.

C. Objectives

- To utilize the Python programming language in replicating the Pacman game, especially its object-oriented programming.
- To implement different concepts of the Python language in replicating the game.
- To provide a unique new experience in playing the game Pacman.
- To design a multiverse-like themed maze, with different game universes serving as the other textures.

D. Significance of the Project

The Pacman game has been in the gaming industry for over numerous decades and has been one of the longest running games that serves as a foundation to many other games at present. However, further experimentation and research will render a better output of the game and will also allow a smoother game experience. This project will be a positive contribution to the education and sector, especially to both teachers and students who are taking up programming or coding courses. The creation of a modified Pacman game in Python could aid other programmers in developing games in the Python programming language.

This study could greatly contribute to game developers in their creation of Python games. The findings of this study could aid game developers in programming games in Python language that would integrate improved game design, quality, and purpose. The findings could be a foundation or reference towards designing a similar game or gain inspiration on the codes used and thus, improve the programming sector with more knowledge about Python game programming.

II. Review of Related Literature

The popularity of games is increasing and attracting numerous people especially the teenagers thanks to its many features; from multimedia opportunities to its educational purposes. Play comprises an intense learning experience in which the participants voluntarily invest a lot of time, energy and commitment, while concurrently deriving great enjoyment from the overall experience (Rieber et al.,1998). Majority of entertainment and recreational activities can be seen in the form of digital games, or video games, which are interactive games that require its games system to integrate technology. As technology continues to evolve, video games have become one of the many ways that the Internet has changed how a generation of young people socialize and view entertainment (Annetta, 2008). Games have varying features and designs that have become difficult to identify from one another. Different game types would have varying impacts, including addictive potential, underlying motives, and mobilization of distinct cognitive processes (Deleuze et al., 2017). Games could be used in a variety of ways, and one of which is as an educational tool.

Nowadays, there are several contests held annually where contestants involve different games like Tetris, Super Mario, StarCraft, Unreal Tournament, Pac-Man, and General Video Game Playing. Pac-Man is a Namco maze action game that was first released in arcades in 1980. The original Japanese name of PuckMan was changed to Pac-Man for international distribution in order to prevent the harm caused by arcade machines. Pacman games can teach the brain to think quickly are the focus of this study. The four ghost enemies in the original Pac-Man game each have a distinct objective to defeat the player. The player must dodge the first ghost who acts as a striker since it will be trying to find the quickest route to them. The second ghost's job is to block the road's course while avoiding the closest player. The third ghost attempts to stop players from utilizing the tunnel on his side and is also in charge of the intercept in the middle of the maze. The fourth ghost, who is merely wandering aimlessly about the game's conclusion, is keeping the player from winning. (Wulandari & Harnadi, 2014). The game often uses a player’s cognitive abilities, in which the player must pay close attention, make quick decisions, and use strategy and timing in order to dodge the enemy's pursuit. Although

the gameplay and aesthetics of this game are straightforward, it may nevertheless pique players' interest, particularly when they try to figure out the best way to use their food without being eaten by the enemy. The player can get knowledge on how to choose a strategy in gaming, mental agility, and focus.

Digital gaming has the potential to improve instruction, as seen by the growing interest in using games in education. Digital games may enhance teaching, student learning and performance, as well as help students build the skills they need to succeed in the twenty-first century. Games are defined by a clear set of rules, and a gamer's performance in playing them is typically explicitly determined by the game's score or outcome. As such, games serve as an excellent proving ground for new methodologies and technologies in academia (Rohlfshagen et al., 2018). Scientific research has long used games as a common testbed, especially in the field of computational intelligence, but also in areas such as psychology, where games might be a method to investigate certain impacts on human test participants.

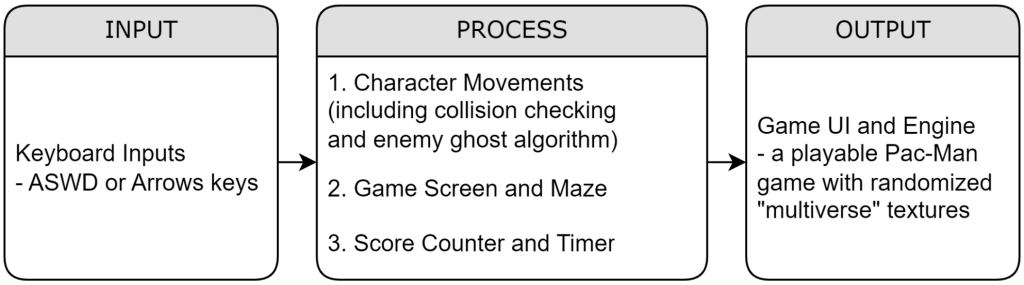
III. Methodology

The project made use of the base concepts of the Python programming language combined with the library Pygame, which aided the program in executing the necessary mechanics and displaying the game. Furthermore, online guides, like those for recreating the game (Jileček, 2022; Liu, 2019), aided the proponents in building the program. The flow diagram that follows explains briefly how the program will function.

A. Conceptual Framework

The conceptual framework or Input-Process-Output chart below shows the different user inputs, processes, and program outputs that make up the framework.

Figure 1: Input-Process-Output (IPO) Diagram



The user inputs include the ASWD and Arrow keys, which allow the player to control the direction of Pac-Man in the game. Meanwhile, There are three superset of processes that the program is built upon: character movement, game screen and maze, and score counter and time.

The character movement processes make sure that the characters, Pac-Man and the ghosts, are able to move inside the game. In this process, Pac-Man follows the direction the player is controlling it to, while the ghosts have two algorithms: scatter randomly or chase Pac-man. In the former, the ghosts will pinpoint a random free space in the maze and determine the most optimal path to get there, which it will then take. On the other hand, the chase mode of the ghosts will pinpoint the location of Pac-Man, go to that position, and repeat.

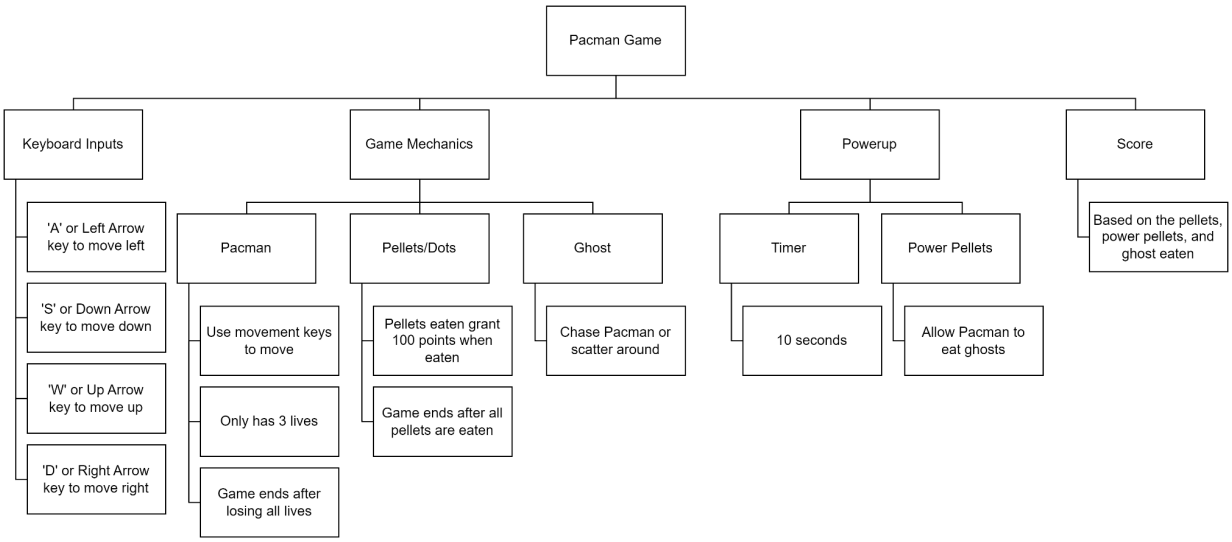
Next, the game screen and maze processes render the game. The game screens make sure that the screen and the characters have a unified unit size, while the maze process renders the walls, pellets, power pellets, and characters of the game. These will also make sure that the right images will be loaded into the characters and the wall. There are seven (7) different folders of assets that the game can use to display the multiverse theme of the project. These assets are themed from six different games excluding Pac-Man, which are: Super Mario Bros, The Legend of Zelda, Sonic, Pokemon, Kirby, and Megaman.

Lastly, the score counter is self-explanatory, as it counts the score of the player. The player gains 100 points per pellet, 250 points per power pellet, and 500 per ghost or creature that were eaten.

All these inputs and processes produce the output — the working replication of the Pac-Man game, with a twist of “multiverse madness.”

B. Hierarchy Chart

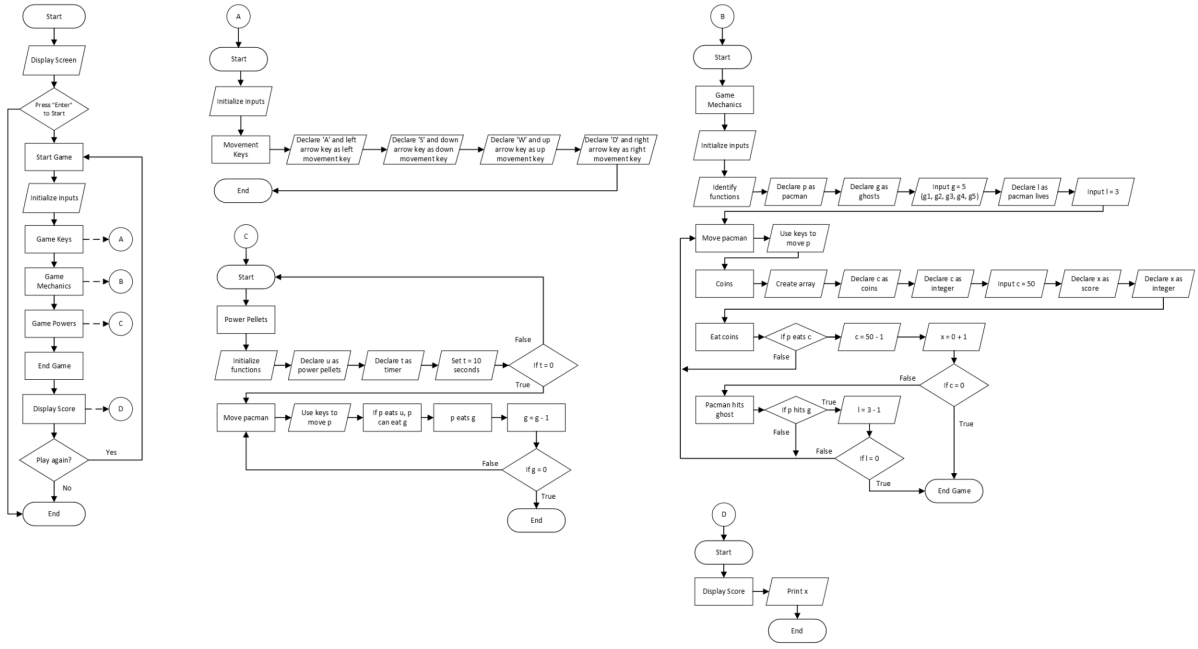
Figure 2: Hierarchy of Modules/Processes



C. Flowchart

The leftmost flowchart represents the overall system of the game, where some variables named as letters, A, B, C, and D are expanded and explained further by the corresponding flowcharts on the right. The ‘A’ flowchart represents the game keys that are initialized to play the game, this includes the movement keys. The ‘B’ flowchart showcases the game mechanics, and determines how to win or lose the game, while the ‘C’ flowchart incorporates the extra options or the game powers that help the player win the game. The ‘D’ flowchart displays the player’s game score.

Figure 3: Program/System Flowchart



D. Gantt Chart

Table 1: Gantt Chart

Activity	Team Member Assigned	Month	March				April			
		Week	1	2	3	4	1	2	3	4
		Day	26-4	5-11	12-18	19-25	26-1	2-8	9-15	17-21
Initial Project Proposal	Tabanao, Tiu		xxxx	xxxx						
Final Project Proposal	Tabanao, Tiu				xxxx	xx				
Game Mechanics Development	Tabanao, Tiu					xx	xxxx	xxxx		
Game Coding	Tabanao, Tiu						xxxx	xxxx	xxxx	
Game Design	Tabanao, Tiu						xxxx	xxxx	xxxx	
Project/ Document Review	Tabanao, Tiu								xxxx	xx
Demonstration	Tabanao, Tiu									xx

IV. Results

Figure 4: The Game with the Original Pac-Man Theme

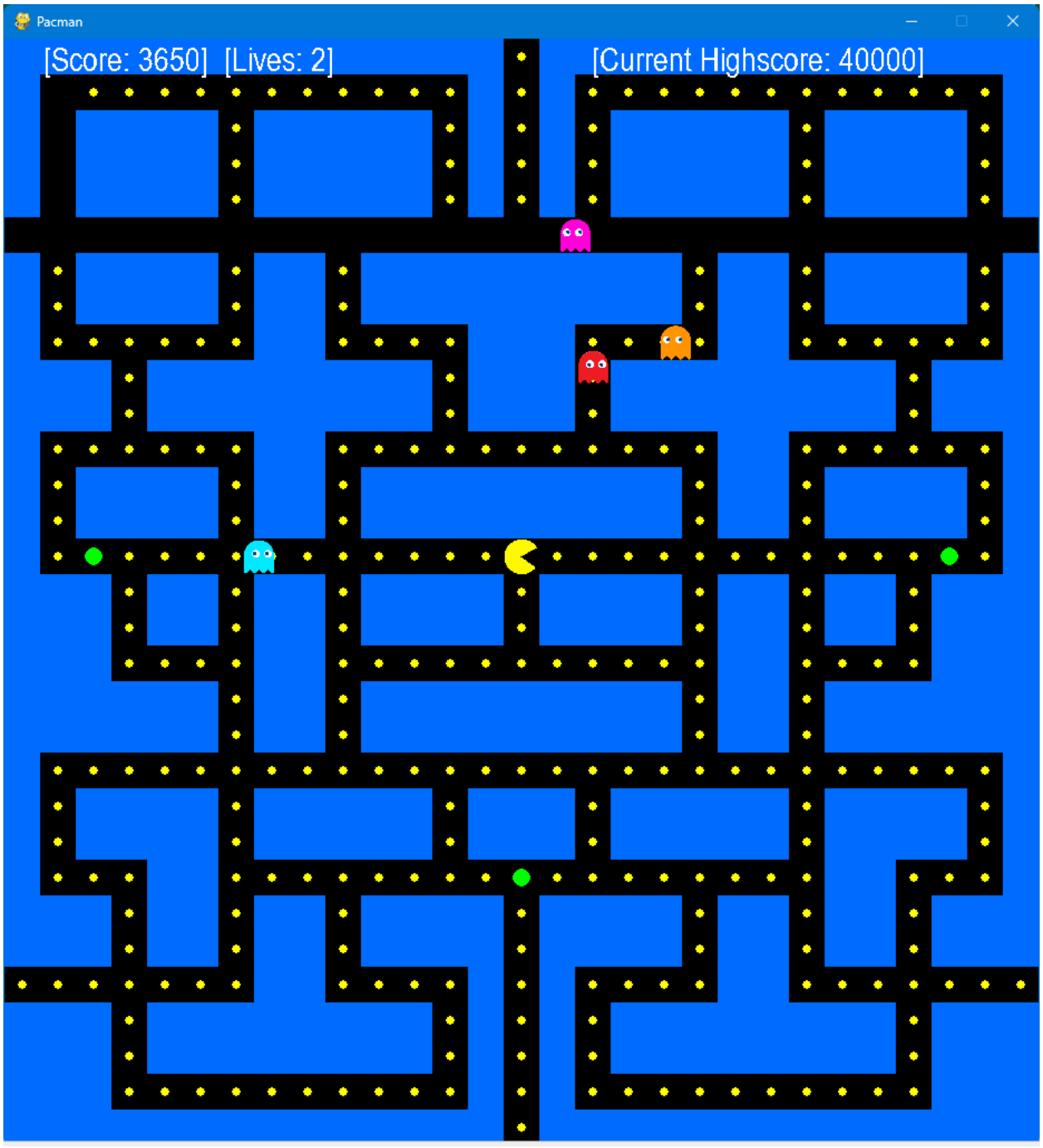


Figure 5: The Game with Super Mario Bros. Theme

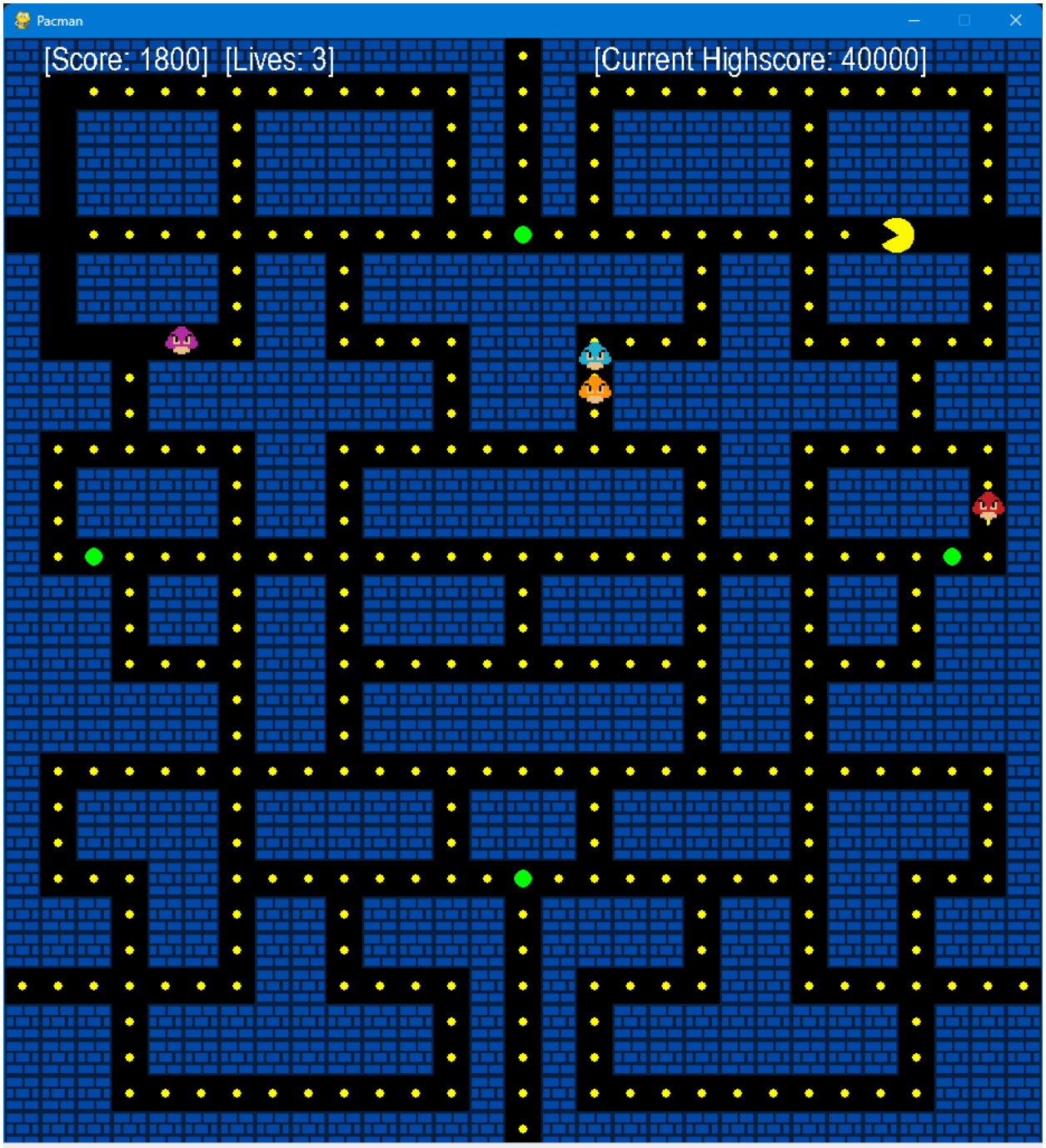


Figure 6: The Game with The Legend of Zelda Theme

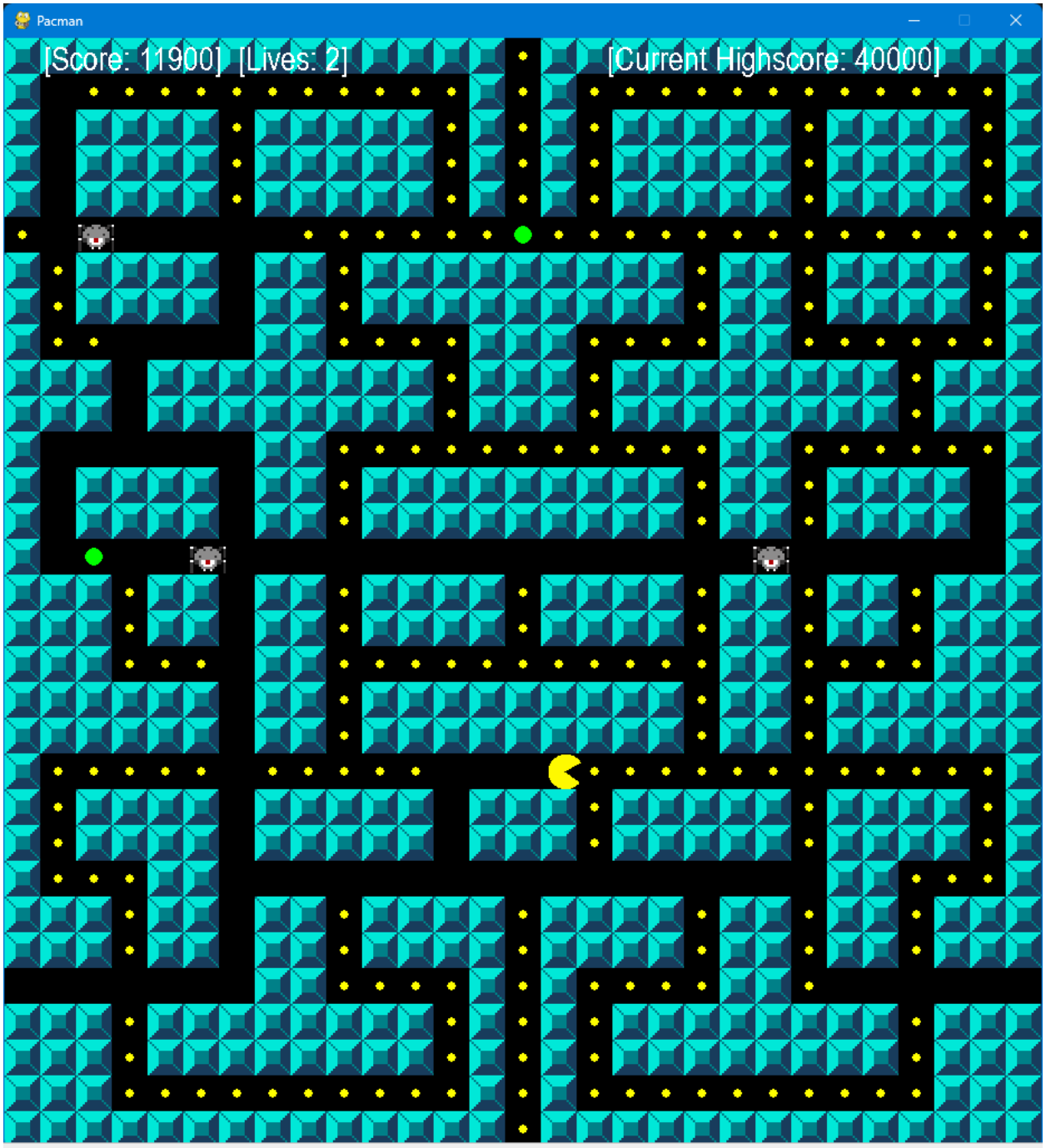


Figure 7: The Game with Kirby Theme

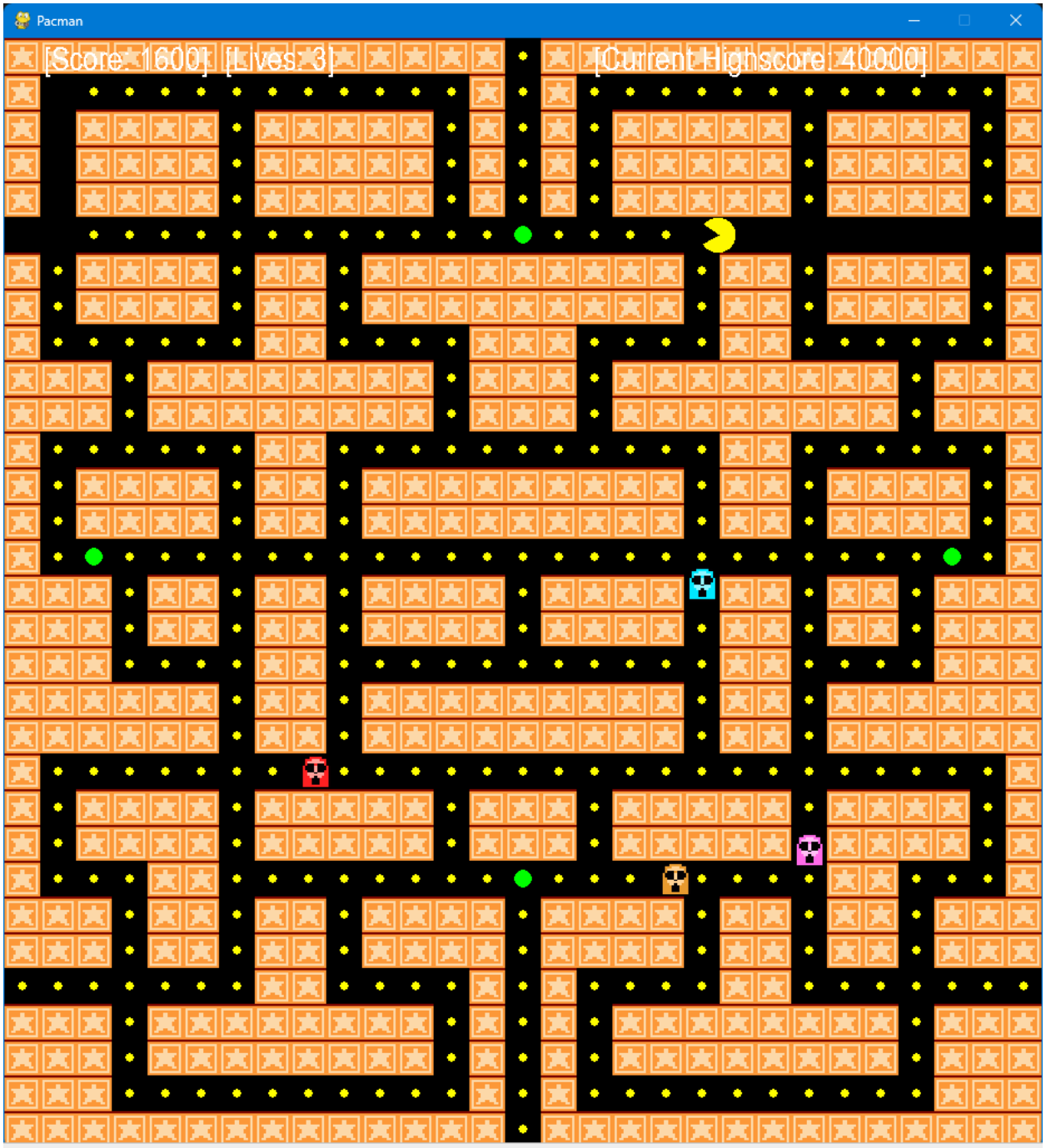


Figure 8: The Game with Pokemon Theme

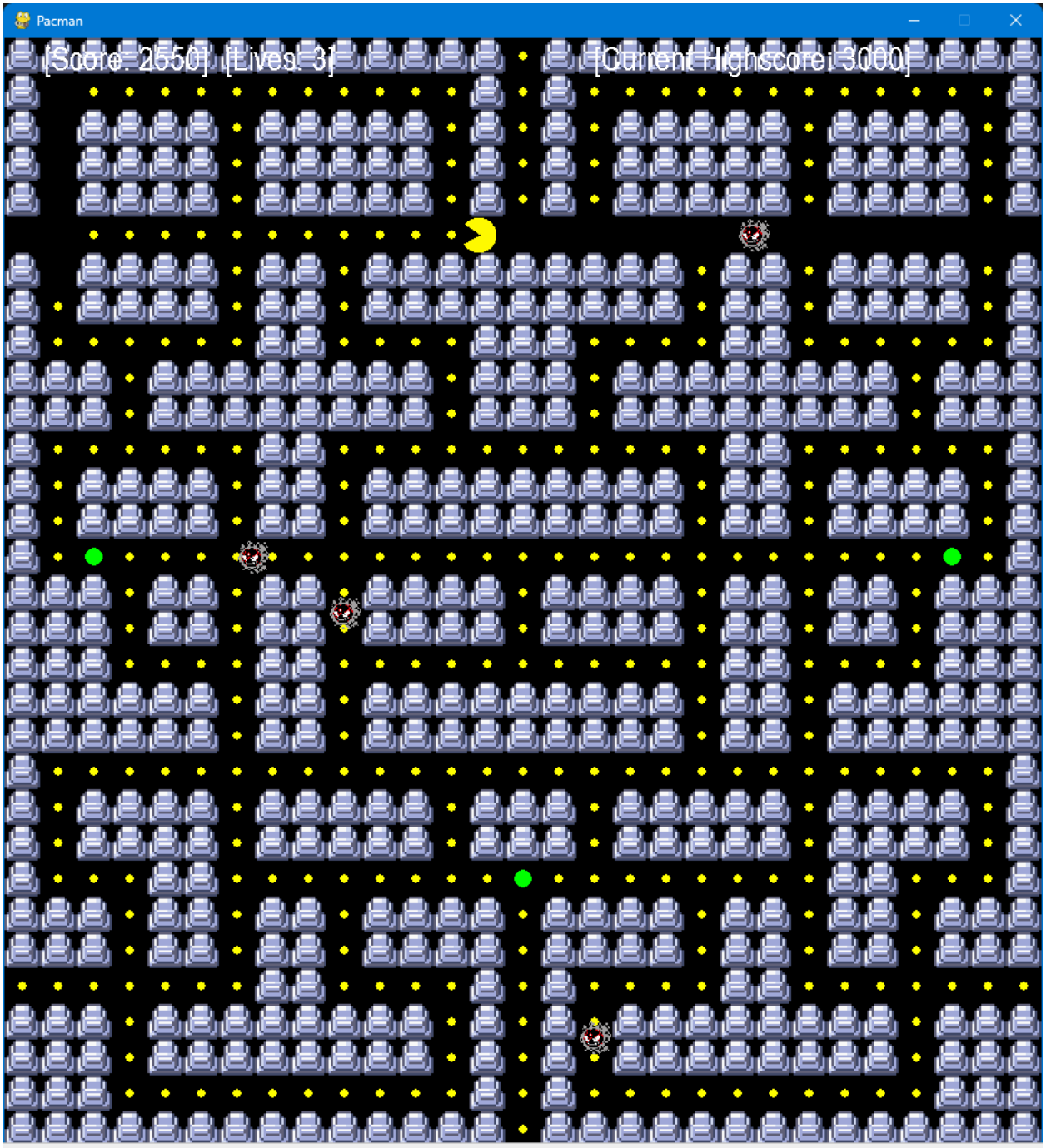


Figure 9: The Game with Mega Man Theme

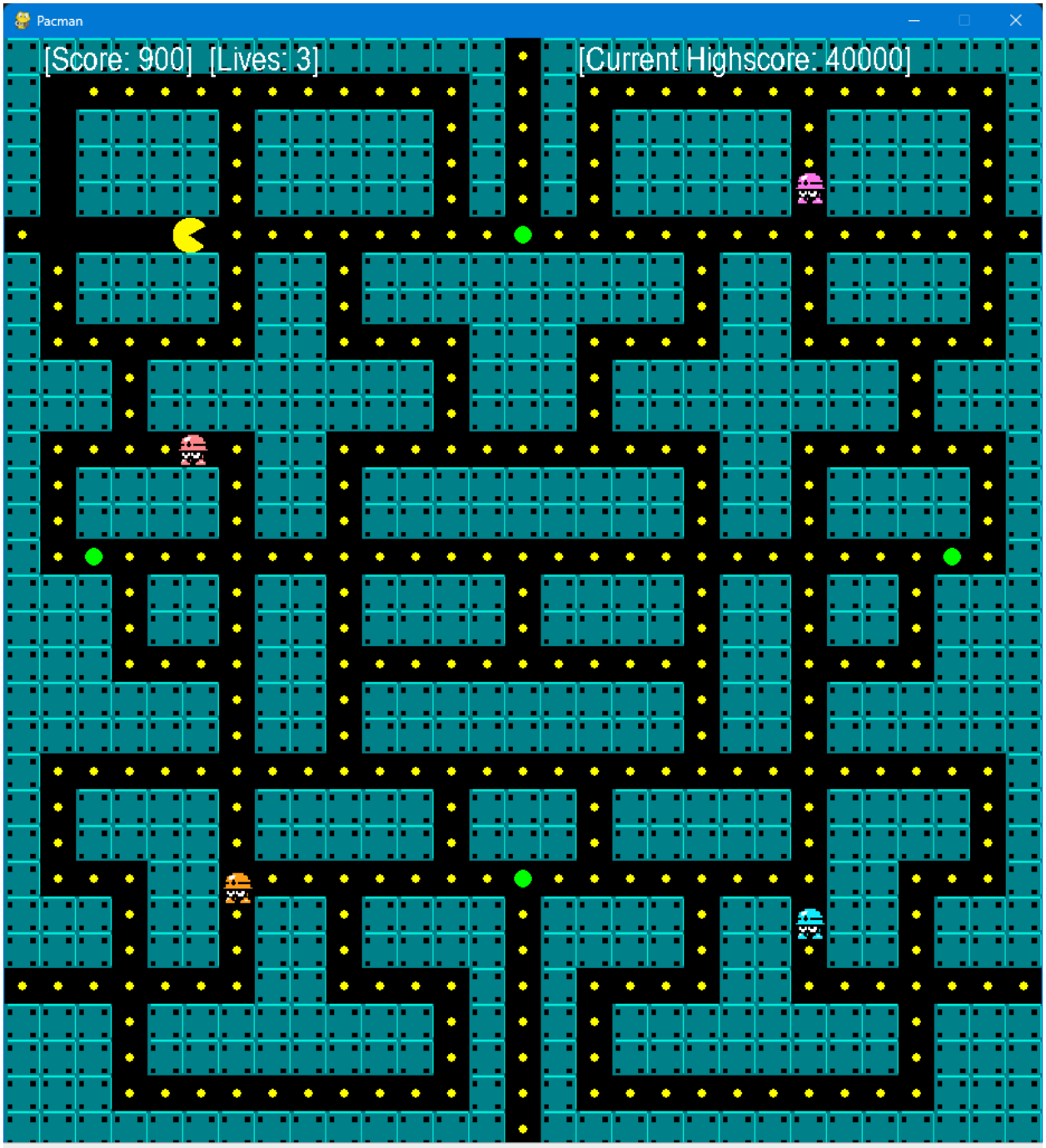


Figure 10: The Game with Sonic the Hedgehog Theme

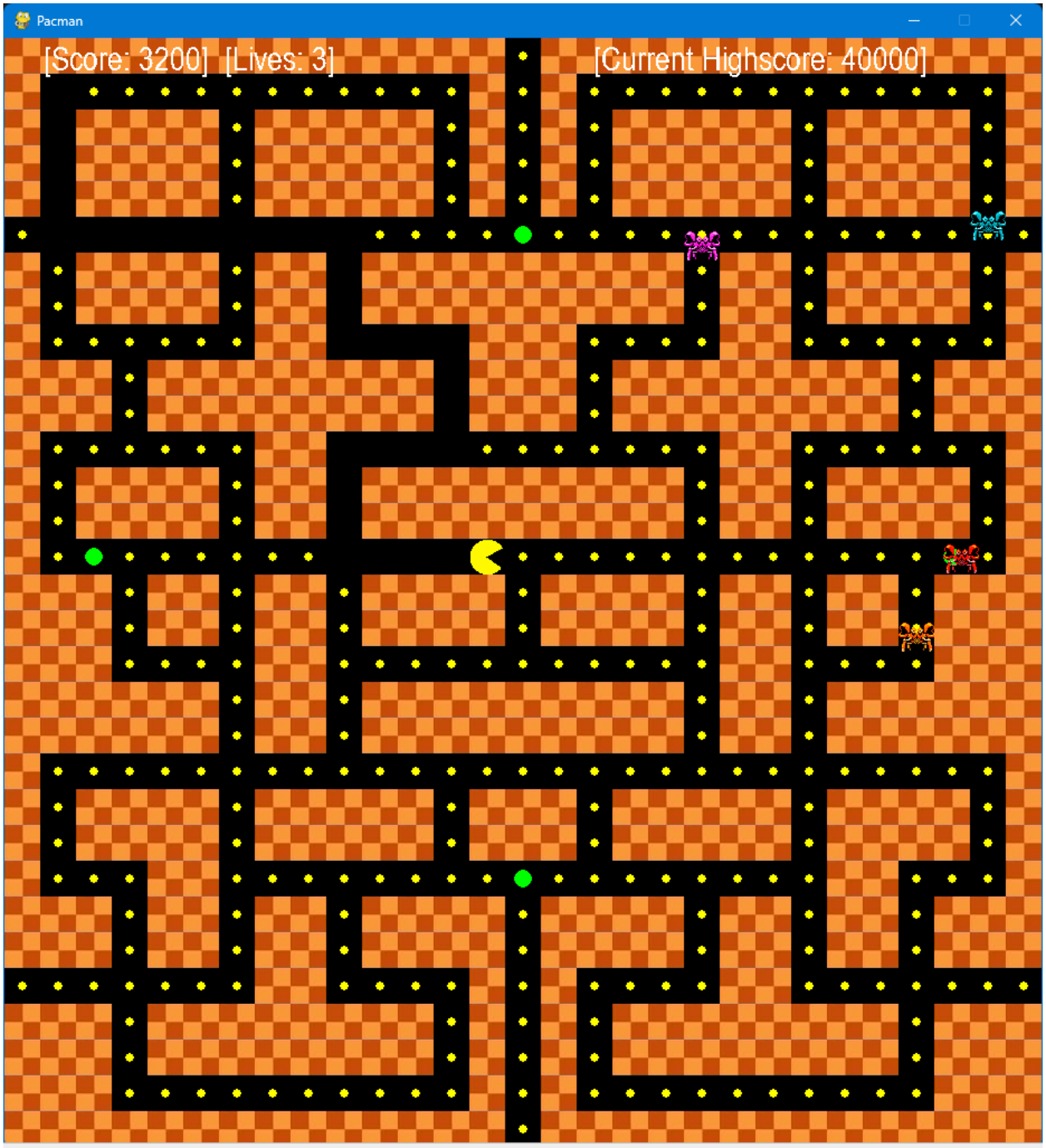


Figure 11.1: Behind the Scenes and Game Over Details (Player Wins with new Highest Score)

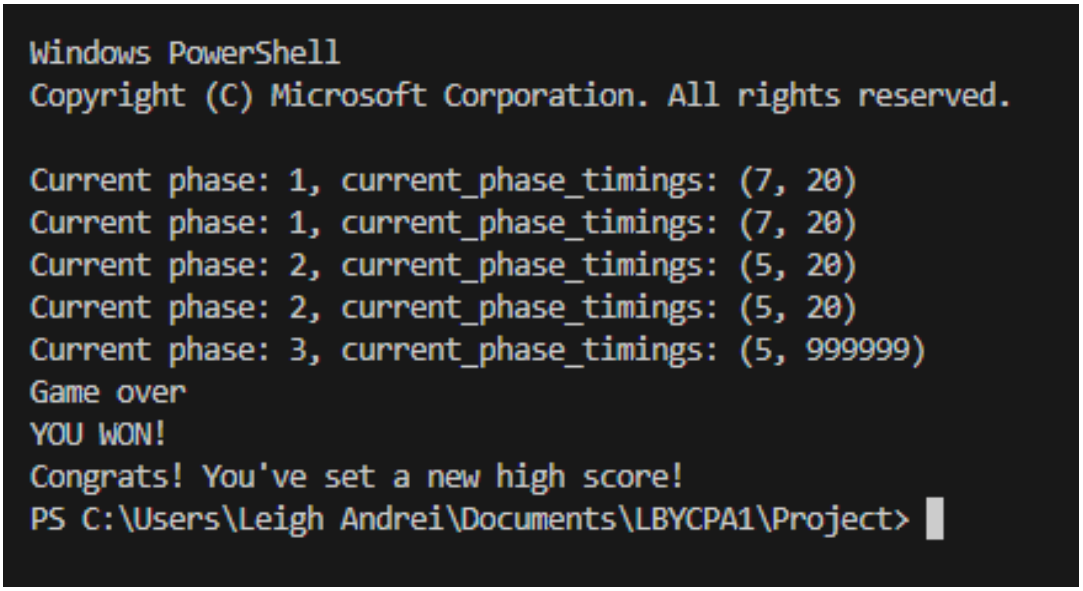


Figure 11.2: Behind the Scenes and Game Over Details (Player Wins)

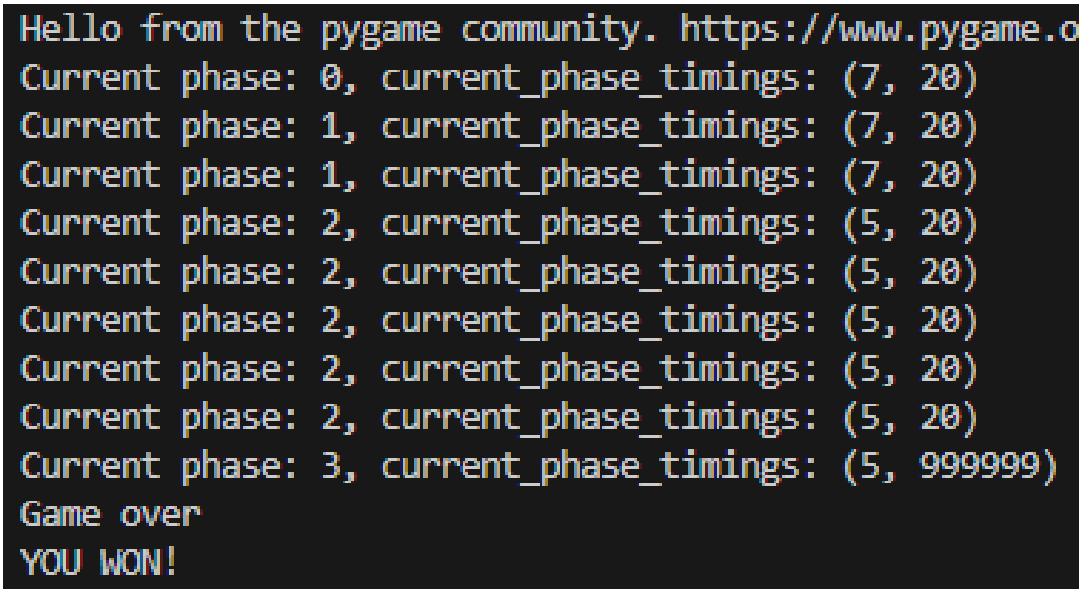
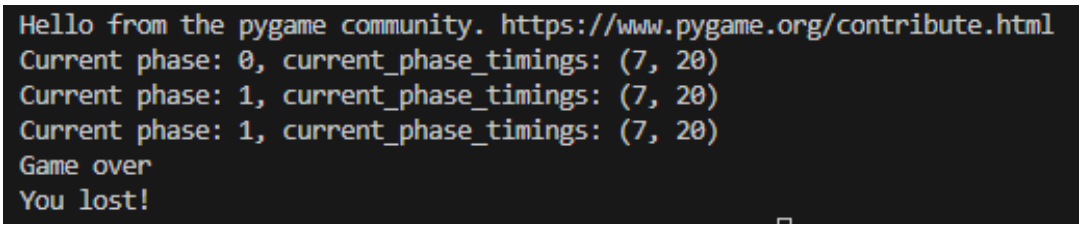


Figure 11.3: Behind the Scenes and Game Over Details (Player Loses)



V. Discussion of Results

The results above show the various game windows that the player may be able to play with our program. Though the game is similar to the regular Pacman game, it still deviates in some areas.

For starters, the theme of the game, particularly the walls and the enemies, switches randomly to seven different games, including Super Mario Bros, The Legend of Zelda, Kirby, Pokemon, Mega Man, and Sonic the Hedgehog. These are illustrated in Figures 4 to 10, in the exact same order.

As what can be seen from the images, the map of the maze has also been changed. Additionally, there are added teleportation routes for Pacman, giving the player more options to outmaneuver the enemies. The ghosts or enemies also do not respawn. However, to compensate for this, Pacman has been nerfed — he now stops for a bit when trying to change direction. Players must now take this into account when trying to evade the enemies in the maze.

Figures 11.1, 11.2, and 11.3 showcases the "behind the scenes" of the game — the different phases of the game — and the game over menu, which is displayed on the terminal of the programming application used.

VI. Analysis, Conclusion and Future Directives

Gaming, particularly of the electronic nature, have quickly become one of the go-to entertainment methods in the modern world. This has also spawned many different technological innovations throughout the recent years. Many of these games have also started incorporating critical thinking and concentration in their design, allowing people to practice these skills even during their leisure time. This shows the evolution of the gaming industry that has attracted audiences from all over the world, inclusive of any age group.

For many different equipment and gadgets, Pacman is available in numerous variations, which was possible because of its simple yet unique mechanics. This has also allowed the students to replicate the game using the Python programming language. The proponents were also able to add their own multiversal twist to the game. This allows the game to have simple but heartwarming surprises for those who will play the game.

However, the proponents failed to implement some of the initially planned features of the game such as the inclusion of either a screen-scrolling feature or a multi-screen maze. To further the development of the program, future projects may include these left-out features in addition to implementing a multi-level system, just like in the original game. Similarly, multiple maze designs may also be implemented to these levels, wherein randomness can also be factored in so that the levels are different for every game. Lastly, two other screens may be added as a game menu and a game over display, this will allow the program to become a more fully-fledged game.

Overall, the game produced by the students for this project can be a good starting point for beginner-programmers who would like to experiment on the mechanics and design aspect of the original Pacman games. It can also provide entertainment to some gamers who would like to play a fresh new version of the game to relax for a bit. The game may not be the best, but it can surely provide some amusement, particularly to casual gamers.

References

Annetta, L.A. (2008). Video Games in Education: Why They Should Be Used and How They Are Being Used. *Theory Into Practice*, *47*(3), 229–239. https://doi.org/10.1080/00405840802153940

Deleuze, J., Christiaens, M., Nuyens, F., & Billieux, J. (2017). Shoot at first sight! First person shooter players display reduced reaction time and compromised inhibitory control in comparison to other video game players. *Computers in Human Behavior*, *72*, 570–576. https://doi.org/10.1016/j.chb.2017.02.027

Jileček, I. J. (2022). Creating a Pac-Man clone in Python in 300 lines of code or less. *ITNEXT - Medium.* https://itnext.io/how-to-create-pac-man-in-python-in-300-lines-of-code-or-less-part-1-288d54baf939

Liu, A. T. (2019). Pacman With AI Python. [GitHub Documentation/ Repository] https://github.com/andi611/Pacman-With-AI-Python/blob/master/task1_Search/pacman.py

Rieber, L.P., Smith, L.B., & Noah, D.I. (1998). The Value of Serious Play. *Educational Technology archive, 38*, 29-37.

Rohlfshagen, P., Liu, J., Perez-Liebana, D., & Lucas, S. M. (2018). Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. *IEEE Transactions on Games*, *10*(3), 233–256. https://doi.org/10.1109/tg.2017.2737145

Vayadande, K., More, H., More, O., Mulay, S., Pathak, A., & Talnikar, V. (2022). Pac Man: Game Development using PDA and OOP. *IRJET Journal*, *09*(01), e-ISSN: 2395-0056. https://www.irjet.net

Wulandari, W., & Harnadi, B. (2014). Pacman Game Benefit Analysis on Decision Making Speed. *SISFORMA*. https://doi.org/10.24167/sisforma.v1i1.88

Appendices

A. User's Manual

To begin, download the necessary Python libraries needed to run the game. These installations can be done using the windows command prompt and pip, a program that comes pre-installed when Python is installed. If the Python language has not been installed, install via the Windows App Store or from www.python.org. To check if Python is installed, open command prompt and type in the following:

```
python - version
```

To check if pip is also installed, type in:

```
pip -version
```

Here are the keywords to install the necessary libraries:

```
pip install pygame
```

```
pip install numpy
```

```
pip install tcod
```

```
pip install random
```

```
pip install enum
```

```
pip install re
```

```
pip install sys
```

After installing all these Python libraries, download a programming application if you do not have one. We recommend using Visual Studio Code by the Microsoft team. Then, download the game files from the GitHub repository. To download, just click on the green "< > Code" button, Lastly, open the file multipacman.py in a programming application. Run the program, then play the game!

B. Source Code

```
import pygame
import numpy as np
import tcod
import random
from enum import Enum
import re
import sys

theme = random.randint(0, 255)
match(theme%8):
    case 0:
        AssetPath = "Assets/Pacman/"

    case 1:
        AssetPath = "Assets/Mario/"

    case 2:
        AssetPath = "Assets/Zelda/"

    case 3:
        AssetPath = "Assets/Kirby/"

    case 4:
        AssetPath = "Assets/Pokemon/"

    case 5:
        AssetPath = "Assets/Megaman/"

    case 6:
        AssetPath = "Assets/Sonic/"

    case 7:
        AssetPath = "Assets/Pacman/"

with open("highscore.txt", "r") as file_highscore:
    current_highscore = int(file_highscore.read())

class Direction(Enum):
    DOWN = -90
    RIGHT = 0
    UP = 90
    LEFT = 180
    NONE = 360
```

```
class ScoreType(Enum):
    COOKIE = 100
    POWERUP = 250
    GHOST = 1000


class GhostBehaviour(Enum):
    CHASE = 1
    RANDOM = 2


def translate_screen_to_maze(in_coords, in_size=32):
    return int(in_coords[0] / in_size), int(in_coords[1] / in_size)


def translate_maze_to_screen(in_coords, in_size=32):
    return in_coords[0] * in_size, in_coords[1] * in_size


class GameObject:
    def __init__(self, in_surface, x, y,
                 in_size: int, in_color=(255, 0, 0),
                 is_circle: bool = False):
        self._size = in_size
        self._renderer: GameRenderer = in_surface
        self._surface = in_surface._screen
        self.y = y
        self.x = x
        self._color = in_color
        self._circle = is_circle
        self._shape = pygame.Rect(self.x, self.y, in_size, in_size)

    def draw(self):
        if self._circle:
            pygame.draw.circle(self._surface,
                               self._color,
                               (self.x, self.y),
                               self._size)
        else:
            rect_object = pygame.Rect(self.x, self.y, self._size, self._size)
            pygame.draw.rect(self._surface,
                             self._color,
                             rect_object,
                             border_radius=1)

    def tick(self):
```

```
        pass

    def get_shape(self):
        return pygame.Rect(self.x, self.y, self._size, self._size)

    def set_position(self, in_x, in_y):
        self.x = in_x
        self.y = in_y

    def get_position(self):
        return (self.x, self.y)


class Wall(GameObject):
    def __init__(self, in_surface, x, y, in_size: int, in_color=(0, 0, 255)):
        super().__init__(in_surface, x * in_size, y * in_size, in_size, in_color)
        self.image = pygame.image.load(AssetPath + "wall.png")

    def draw(self):
        self.image = pygame.transform.scale(self.image, (32, 32))
        self._surface.blit(self.image, self.get_shape())


class GameRenderer:
    def __init__(self, in_width: int, in_height: int):
        pygame.init()
        self._width = in_width
        self._height = in_height
        self._screen = pygame.display.set_mode((in_width, in_height))
        pygame.display.set_caption('Pacman')
        self._clock = pygame.time.Clock()
        self._done = False
        self._won = False
        self._game_objects = []
        self._walls = []
        self._cookies = []
        self._powerups = []
        self._ghosts = []
        self._hero: Hero = None
        self._lives = 3
        self._score = 0
        self._score_cookie_pickup = 100
        self._score_ghost_eaten = 1000
        self._score_powerup_pickup = 250
        self._powerup_active = False
        self._current_mode = GhostBehaviour.RANDOM
```

```
        self._mode_switch_event = pygame.USEREVENT + 1
        self._powerup_end_event = pygame.USEREVENT + 2
        self._pacman_event = pygame.USEREVENT + 3
        self._modes = [
            (7, 20),
            (7, 20),
            (5, 20),
            (5, 999999)
        ]
        self._current_phase = 0

    def tick(self, in_fps: int):
        black = (0, 0, 0)

        self.handle_mode_switch()
        pygame.time.set_timer(self._pacman_event, 200)
        while not self._done:
            for game_object in self._game_objects:
                game_object.tick()
                game_object.draw()

            self.display_text(f"     [Score: {self._score}]  [Lives: {self._lives}]
[Current Highscore: {current_highscore}]")

            if self._hero is None or self.get_won():
                print("Game over")
                if self._hero is None: print("You lost!")
                if self.get_won(): print("YOU WON!")
                if self._score > current_highscore:
                    print("Congrats! You've set a new high score!")
                    with open("highscore.txt", "w") as overwrite_highscore:
                        overwrite_highscore.write(str(self._score))
                elif self._score > current_highscore:
                    print("Wow! You got a high score!")
                pygame.quit()
                sys.exit()

            pygame.display.flip()
            self._clock.tick(in_fps)
            self._screen.fill(black)
            self._handle_events()

    def handle_mode_switch(self):
        current_phase_timings = self._modes[self._current_phase]
        print(f"Current phase: {str(self._current_phase)}, current_phase_timings:
{str(current_phase_timings)}")
```

```
        scatter_timing = current_phase_timings[0]
        chase_timing = current_phase_timings[1]

        if self._current_mode == GhostBehaviour.CHASE:
            self._current_phase += 1
            self.set_current_mode(GhostBehaviour.RANDOM)
        else:
            self.set_current_mode(GhostBehaviour.CHASE)

        used_timing = scatter_timing if self._current_mode == GhostBehaviour.RANDOM else
chase_timing
        pygame.time.set_timer(self._mode_switch_event, used_timing * 1000)

    def start_powerup_timeout(self):
        pygame.time.set_timer(self._powerup_end_event, 10000)

    def add_game_object(self, obj: GameObject):
        self._game_objects.append(obj)

    def add_cookie(self, obj: GameObject):
        self._game_objects.append(obj)
        self._cookies.append(obj)

    def add_ghost(self, obj: GameObject):
        self._game_objects.append(obj)
        self._ghosts.append(obj)

    def add_powerup(self, obj: GameObject):
        self._game_objects.append(obj)
        self._powerups.append(obj)

    def activate_powerup(self):
        self._powerup_active = True
        self.set_current_mode(GhostBehaviour.RANDOM)
        self.start_powerup_timeout()

    def set_won(self):
        self._won = True

    def get_won(self):
        return self._won

    def add_score(self, in_score: ScoreType):
        self._score += in_score.value

    def get_hero_position(self):
```

```
    return self._hero.get_position() if self._hero != None else (0, 0)

def set_current_mode(self, in_mode: GhostBehaviour):
    self._current_mode = in_mode

def get_current_mode(self):
    return self._current_mode

def end_game(self):
    if self._hero in self._game_objects:
        self._game_objects.remove(self._hero)
    self._hero = None

def kill_pacman(self):
    self._lives -= 1
    self._hero.set_position(32 * 14, 32 *14)
    self._hero.set_direction(Direction.NONE)
    if self._lives == 0: self.end_game()

def display_text(self, text, in_position=(0, 0), in_size=30):
    font = pygame.font.SysFont('Arial', in_size)
    text_surface = font.render(text, False, (255, 255, 255))
    self._screen.blit(text_surface, in_position)

def is_powerup_active(self):
    return self._powerup_active

def add_wall(self, obj: Wall):
    self.add_game_object(obj)
    self._walls.append(obj)

def get_walls(self):
    return self._walls

def get_cookies(self):
    return self._cookies

def get_ghosts(self):
    return self._ghosts

def get_powerups(self):
    return self._powerups

def get_game_objects(self):
    return self._game_objects
```

```
    def add_hero(self, in_hero):
        self.add_game_object(in_hero)
        self._hero = in_hero

    def _handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self._done = True

            if event.type == self._mode_switch_event:
                self.handle_mode_switch()

            if event.type == self._powerup_end_event:
                self._powerup_active = False

            if event.type == self._pacman_event:
                if self._hero is None: break
                self._hero.mouth_open = not self._hero.mouth_open

        pressed = pygame.key.get_pressed()
        if self._hero is None: return
        if pressed[pygame.K_m]:
            AssetPath = "Assets/Mario/"
        if pressed[pygame.K_UP] or pressed[pygame.K_w]:
            self._hero.set_direction(Direction.UP)
        elif pressed[pygame.K_LEFT] or pressed[pygame.K_a]:
            self._hero.set_direction(Direction.LEFT)
        elif pressed[pygame.K_DOWN] or pressed[pygame.K_s]:
            self._hero.set_direction(Direction.DOWN)
        elif pressed[pygame.K_RIGHT] or pressed[pygame.K_d]:
            self._hero.set_direction(Direction.RIGHT)


class MovableObject(GameObject):
    def __init__(self, in_surface, x, y, in_size: int, in_color=(255, 255, 0), is_circle: bool =
False):
        super().__init__(in_surface, x, y, in_size, in_color, is_circle)
        self.current_direction = Direction.NONE
        self.direction_buffer = Direction.NONE
        self.last_working_direction = Direction.NONE
        self.location_queue = []
        self.next_target = None
        self.image = pygame.image.load(AssetPath + 'ghost.png')

    def get_next_location(self):
        return None if len(self.location_queue) == 0 else self.location_queue.pop(0)
```

```
    def set_direction(self, in_direction):
        self.current_direction = in_direction
        self.direction_buffer = in_direction

    def collides_with_wall(self, in_position):
        collision_rect = pygame.Rect(in_position[0], in_position[1], self._size, self._size)
        collides = False
        walls = self._renderer.get_walls()
        for wall in walls:
            collides = collision_rect.colliderect(wall.get_shape())
            if collides: break
        return collides

    def check_collision_in_direction(self, in_direction: Direction):
        desired_position = (0, 0)
        if in_direction == Direction.NONE: return False, desired_position
        if in_direction == Direction.UP:
            desired_position = (self.x, self.y - 2)
        elif in_direction == Direction.DOWN:
            desired_position = (self.x, self.y + 2)
        elif in_direction == Direction.LEFT:
            desired_position = (self.x - 2, self.y)
        elif in_direction == Direction.RIGHT:
            desired_position = (self.x + 2, self.y)

        return self.collides_with_wall(desired_position), desired_position

    def auto_move(self, in_direction: Direction):
        pass

    def tick(self):
        self.reached_target()
        self.auto_move(self.current_direction)

    def reached_target(self):
        pass

    def draw(self):
        self.image = pygame.transform.scale(self.image, (32, 32))
        self._surface.blit(self.image, self.get_shape())


class Hero(MovableObject):
    def __init__(self, in_surface, x, y, in_size: int):
        super().__init__(in_surface, x, y, in_size, (255, 255, 0), False)
```

```
        self.last_non_colliding_position = (0, 0)
        self.open = pygame.image.load(AssetPath + "pacman_open.png")
        self.closed = pygame.image.load(AssetPath + "pacman_closed.png")
        self.image = self.open
        self.mouth_open = True


    def tick(self):
        if self.x < 0:
            self.x = self._renderer._width

        if self.x > self._renderer._width:
            self.x = 0

        if self.y < 0:
            self.y = self._renderer._height

        if self.y > self._renderer._height:
            self.y = 0

        self.last_non_colliding_position = self.get_position()

        if self.check_collision_in_direction(self.direction_buffer)[0]:
            self.auto_move(self.current_direction)
        else:
            self.auto_move(self.direction_buffer)
            self.current_direction = self.direction_buffer

        if self.collides_with_wall((self.x, self.y)):
            self.set_position(self.last_non_colliding_position[0],
self.last_non_colliding_position[1])

        self.handle_cookie_pickup()
        self.handle_ghosts()

    def auto_move(self, in_direction: Direction):
        collision_result = self.check_collision_in_direction(in_direction)

        desired_position_collides = collision_result[0]
        if not desired_position_collides:
            self.last_working_direction = self.current_direction
            desired_position = collision_result[1]
            self.set_position(desired_position[0], desired_position[1])
        else:
            self.current_direction = self.last_working_direction
```

```
def handle_cookie_pickup(self):
    collision_rect = pygame.Rect(self.x, self.y, self._size, self._size)
    cookies = self._renderer.get_cookies()
    powerups = self._renderer.get_powerups()
    game_objects = self._renderer.get_game_objects()
    cookie_to_remove = None
    for cookie in cookies:
        collides = collision_rect.colliderect(cookie.get_shape())
        if collides and cookie in game_objects:
            game_objects.remove(cookie)
            self._renderer.add_score(ScoreType.COOKIE)
            cookie_to_remove = cookie

    if cookie_to_remove is not None:
        cookies.remove(cookie_to_remove)

    if len(self._renderer.get_cookies()) == 0:
        self._renderer.set_won()

    for powerup in powerups:
        collides = collision_rect.colliderect(powerup.get_shape())
        if collides and powerup in game_objects:
            if not self._renderer.is_powerup_active():
                game_objects.remove(powerup)
                self._renderer.add_score(ScoreType.POWERUP)
                self._renderer.activate_powerup()

def handle_ghosts(self):
    collision_rect = pygame.Rect(self.x, self.y, self._size, self._size)
    ghosts = self._renderer.get_ghosts()
    game_objects = self._renderer.get_game_objects()
    for ghost in ghosts:
        collides = collision_rect.colliderect(ghost.get_shape())
        if collides and ghost in game_objects:
            if self._renderer.is_powerup_active():
                game_objects.remove(ghost)
                self._renderer.add_score(ScoreType.GHOST)
            else:
                if not self._renderer.get_won():
                    self._renderer.kill_pacman()

def draw(self):
    half_size = self._size / 2
    self.image = self.open if self.mouth_open else self.closed
    self.image = pygame.transform.rotate(self.image, self.current_direction.value)
    super(Hero, self).draw()
```

```
class Ghost(MovableObject):
    def __init__(self, in_surface, x, y, in_size: int, in_game_controller,
sprite_path=AssetPath+"ghost_fright.png"):
        super().__init__(in_surface, x, y, in_size)
        self.game_controller = in_game_controller
        self.sprite_normal = pygame.image.load(sprite_path)
        self.sprite_fright = pygame.image.load(AssetPath + "ghost_fright.png")

    def reached_target(self):
        if (self.x, self.y) == self.next_target:
            self.next_target = self.get_next_location()
        self.current_direction = self.calculate_direction_to_next_target()

    def set_new_path(self, in_path):
        for item in in_path:
            self.location_queue.append(item)
        self.next_target = self.get_next_location()

    def calculate_direction_to_next_target(self) -> Direction:
        if self.next_target is None:
            if self._renderer.get_current_mode() == GhostBehaviour.CHASE and not
self._renderer.is_powerup_active():
                self.request_path_to_player(self)
            else:
                self.game_controller.request_new_random_path(self)
            return Direction.NONE

        diff_x = self.next_target[0] - self.x
        diff_y = self.next_target[1] - self.y
        if diff_x == 0:
            return Direction.DOWN if diff_y > 0 else Direction.UP
        if diff_y == 0:
            return Direction.LEFT if diff_x < 0 else Direction.RIGHT

        if self._renderer.get_current_mode() == GhostBehaviour.CHASE and not
self._renderer.is_powerup_active():
            self.request_path_to_player(self)
        else:
            self.game_controller.request_new_random_path(self)
        return Direction.NONE

    def request_path_to_player(self, in_ghost):
        player_position = translate_screen_to_maze(in_ghost._renderer.get_hero_position())
        current_maze_coord = translate_screen_to_maze(in_ghost.get_position())
```

```
        path = self.game_controller.p.get_path(current_maze_coord[1], current_maze_coord[0],
player_position[1],
                                               player_position[0])

        new_path = [translate_maze_to_screen(item) for item in path]
        in_ghost.set_new_path(new_path)

    def auto_move(self, in_direction: Direction):
        if in_direction == Direction.UP:
            self.set_position(self.x, self.y - 2)
        elif in_direction == Direction.DOWN:
            self.set_position(self.x, self.y + 2)
        elif in_direction == Direction.LEFT:
            self.set_position(self.x - 2, self.y)
        elif in_direction == Direction.RIGHT:
            self.set_position(self.x + 2, self.y)

    def draw(self):
        self.image = self.sprite_fright if self._renderer.is_powerup_active() else
self.sprite_normal
        super(Ghost, self).draw()

class Cookie(GameObject):
    def __init__(self, in_surface, x, y):
        super().__init__(in_surface, x, y, 4, (255, 255, 0), True)

class Powerup(GameObject):
    def __init__(self, in_surface, x, y):
        super().__init__(in_surface, x, y, 8, (0, 255, 0), True)

class Pathfinder:
    def __init__(self, in_arr):
        cost = np.array(in_arr, dtype=np.bool_).tolist()
        self.pf = tcod.path.AStar(cost=cost, diagonal=0)

    def get_path(self, from_x, from_y, to_x, to_y) -> object:
        res = self.pf.get_path(from_x, from_y, to_x, to_y)
        return [(sub[1], sub[0]) for sub in res]


class PacmanGame:
    def __init__(self):
        self.ascii_maze1 = [
            "XXXXXXXXXXXXXX XXXXXXXXXXXXXX",
            "X          X X          X",
            "X XXXX XXXXX X X XXXXX XXXX X",
```

```
        "X XXXX XXXXX X X XXXXX XXXX X",
        "X XXXX XXXXX X X XXXXX XXXX X",
        " G           O           G ",
        "X XXXX XX XXXXXXXXX XX XXXX X",
        "X XXXX XX XXXXXXXXX XX XXXX X",
        "X     XX    XXX    XX     X",
        "XXX XXXXXXXX XXX XXXXXXXX XXX",
        "XXX XXXXXXXX XXX XXXXXXXX XXX",
        "X     XX          XX     X",
        "X XXXX XX XXXXXXXXX XX XXXX X",
        "X XXXX XX XXXXXXXXX XX XXXX X",
        "X O                   O X",
        "XXX XX XX XXXX XXXX XX XX XXX",
        "XXX XX XX XXXX XXXX XX XX XXX",
        "XXX    XX     p     XX    XXX",
        "XXXXXX XX XXXXXXXXX XX XXXXXX",
        "XXXXXX XX XXXXXXXXX XX XXXXXX",
        "X                     X",
        "X XXXX XXXXX XXX XXXXX XXXX X",
        "X XXXX XXXXX XXX XXXXX XXXX X",
        "X   XX       O       XX   X",
        "XXX XX XX XXXX XXXX XX XX XXX",
        "XXX XX XX XXXX XXXX XX XX XXX",
        "       XX    X X    XX       ",
        "XXX XXXXXXXX X X XXXXXXXX XXX",
        "XXX XXXXXXXX X X XXXXXXXX XXX",
        "XXX G       X X       G  XXX",
        "XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX",
    ]

    self.numpy_maze = []
    self.cookie_spaces = []
    self.powerup_spaces = []
    self.reachable_spaces = []
    self.ghost_spawns = []
    self.ghost_colors = [
        AssetPath + "ghost.png",
        AssetPath + "ghost_pink.png",
        AssetPath + "ghost_orange.png",
        AssetPath + "ghost_blue.png"
    ]
    self.size = (0, 0)
    self.convert_maze_to_numpy()
    self.p = Pathfinder(self.numpy_maze)

def request_new_random_path(self, in_ghost: Ghost):
```

```
        random_space = random.choice(self.reachable_spaces)
        current_maze_coord = translate_screen_to_maze(in_ghost.get_position())

        path = self.p.get_path(current_maze_coord[1], current_maze_coord[0],
random_space[1],
                               random_space[0])
        test_path = [translate_maze_to_screen(item) for item in path]
        in_ghost.set_new_path(test_path)

    def convert_maze_to_numpy(self):
        for x, row in enumerate(self.ascii_maze1):
            self.size = (len(row), x + 1)
            binary_row = []
            for y, column in enumerate(row):
                if column == "G":
                    self.ghost_spawns.append((y, x))

                if column == "X":
                    binary_row.append(0)
                else:
                    binary_row.append(1)
                    self.cookie_spaces.append((y, x))
                    self.reachable_spaces.append((y, x))
                    if column == "O":
                        self.powerup_spaces.append((y, x))

            self.numpy_maze.append(binary_row)


if __name__ == "__main__":
    unified_size = 32
    pacman_game = PacmanGame()
    size = pacman_game.size
    game_renderer = GameRenderer(size[0] * unified_size, size[1] * unified_size)

    for y, row in enumerate(pacman_game.numpy_maze):
        for x, column in enumerate(row):
            if column == 0:
                game_renderer.add_wall(Wall(game_renderer, x, y, unified_size))

    for cookie_space in pacman_game.cookie_spaces:
        translated = translate_maze_to_screen(cookie_space)
        cookie = Cookie(game_renderer, translated[0] + unified_size / 2, translated[1] +
unified_size / 2)
        game_renderer.add_cookie(cookie)
```

```
    for powerup_space in pacman_game.powerup_spaces:
        translated = translate_maze_to_screen(powerup_space)
        powerup = Powerup(game_renderer, translated[0] + unified_size / 2, translated[1] +
unified_size / 2)
        game_renderer.add_powerup(powerup)

    for i, ghost_spawn in enumerate(pacman_game.ghost_spawns):
        translated = translate_maze_to_screen(ghost_spawn)
        ghost = Ghost(game_renderer, translated[0], translated[1], unified_size, pacman_game,
                      pacman_game.ghost_colors[i % 4])
        game_renderer.add_ghost(ghost)

    pacman = Hero(game_renderer, unified_size, unified_size, unified_size)
    game_renderer.add_hero(pacman)
    game_renderer.set_current_mode(GhostBehaviour.CHASE)
    game_renderer.tick(120)
```

C. Work breakdown

Student Name	Tasks Assigned	Percentage of Work Contribution
Tabanao, Leigh Andrei	Project Proposal Game Mechanics Development Game Coding Game Design Project/ Document Review Demonstration	50%
Tiu, Timothy Brian	Project Proposal Game Mechanics Development Game Coding Game Design Project/ Document Review Demonstration	50%

D. Personal Data Sheet

Name: Leigh Andrei Tabanao

Student No.: 12217379

Gender: Male

Course: BS Computer Engineering

Email Address: leigh_tabanao@dlsu.edu.ph

Name: Timothy Brian Tiu

Student No.: 12217409

Gender: Male

Course: BS Computer Engineering

Email Address: timothy_brian_tiu@dlsu.edu.ph