| 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 correct, logically  presented and used the required format. | 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 % |
| 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**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

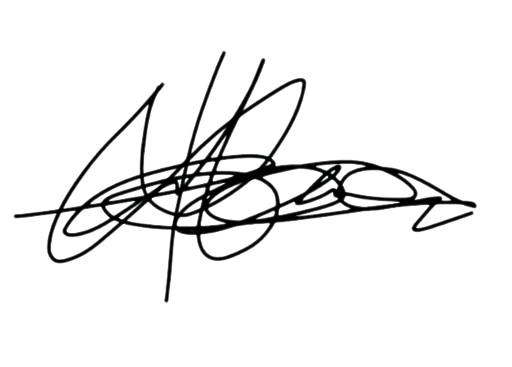
A Term Project

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

[**Table of Contents 5**](#_owmehs1a5abn)

[**List of Figures 6**](#_r0kalgovgrrt)

[**List of Tables 6**](#_720su0xejn6u)

[**I. Introduction 7**](#_1a2yer3npp3a)

[A. Background of the Study 7](#_o8at6lvpjlqb)

[B. Problem Statement 8](#_2tt0rha27j29)

[C. Objectives 8](#_yk8m0p5kmaw4)

[D. Significance of the Project 8](#_5vnup3dshhjr)

[**II. Review of Related Literature 9**](#_te9kquvhee5u)

[**III. Methodology 10**](#_ngf6n97qr70j)

[A. Conceptual Framework 10](#_kiveiuqg9tcx)

[B. Hierarchy Chart 12](#_2drug01avllu)

[C. Flowchart 12](#_to51no84qhyx)

[D. Gantt Chart 13](#_rux44plv3xk2)

[**IV. Results 14**](#_hgqiv0janu14)

[**V. Discussion of Results 22**](#_s409nkur7fq)

[**VI. Analysis, Conclusion and Future Directives 22**](#_p217rghy04ag)

[**References 23**](#_qzsobvqhfvga)

[**Appendices 24**](#_1wrbbfurudol)

[A. User’s Manual 24](#_nojossbxy4nc)

[B. Source Code 25](#_lxwngeauh29o)

[C. Work breakdown 39](#_wjsp9icnvjqx)

[D. Personal Data Sheet 40](#_kwjpf5i1jgmz)

# 

# List of Figures

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

Figure 2: Hierarchy of Modules/Processes

Figure 3: Program/System Flowchart

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)

# List of Tables

Table 1: Gantt Chart

# 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.

## 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.”

## 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.

## 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.

## 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.

# 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.

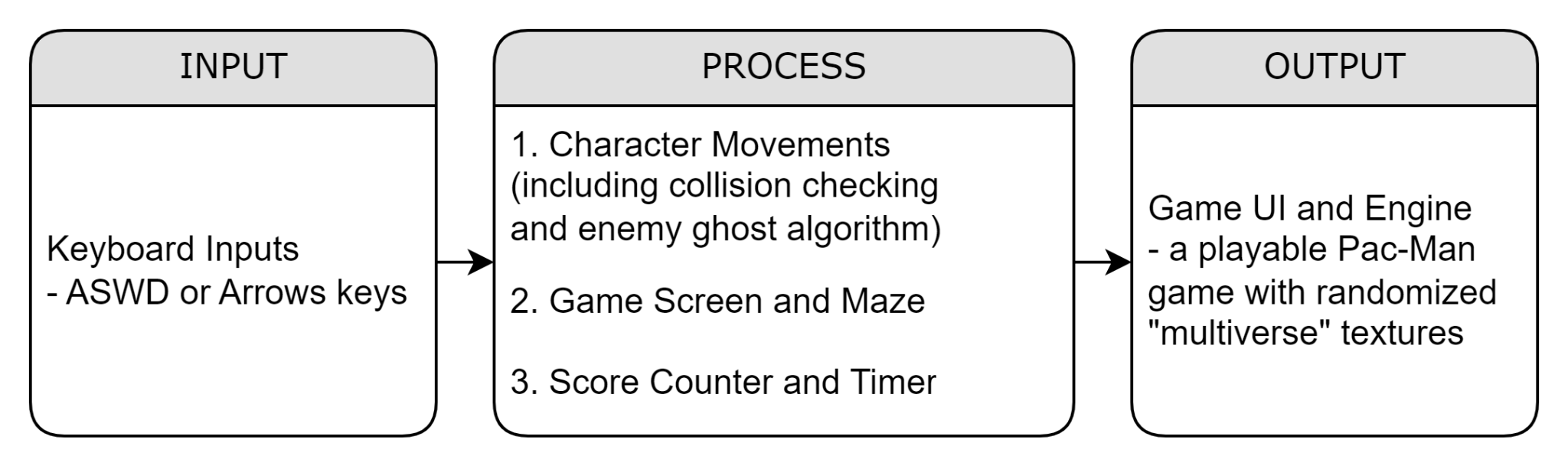
Digital gaming has the potential to improve instruction, as seen by the growing interest in using games in education. Digital games may enhance teaching, student learning and performance, as well as help students build the skills they need to succeed in the twenty-first century. Games are defined by 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.

# 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.

## 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.

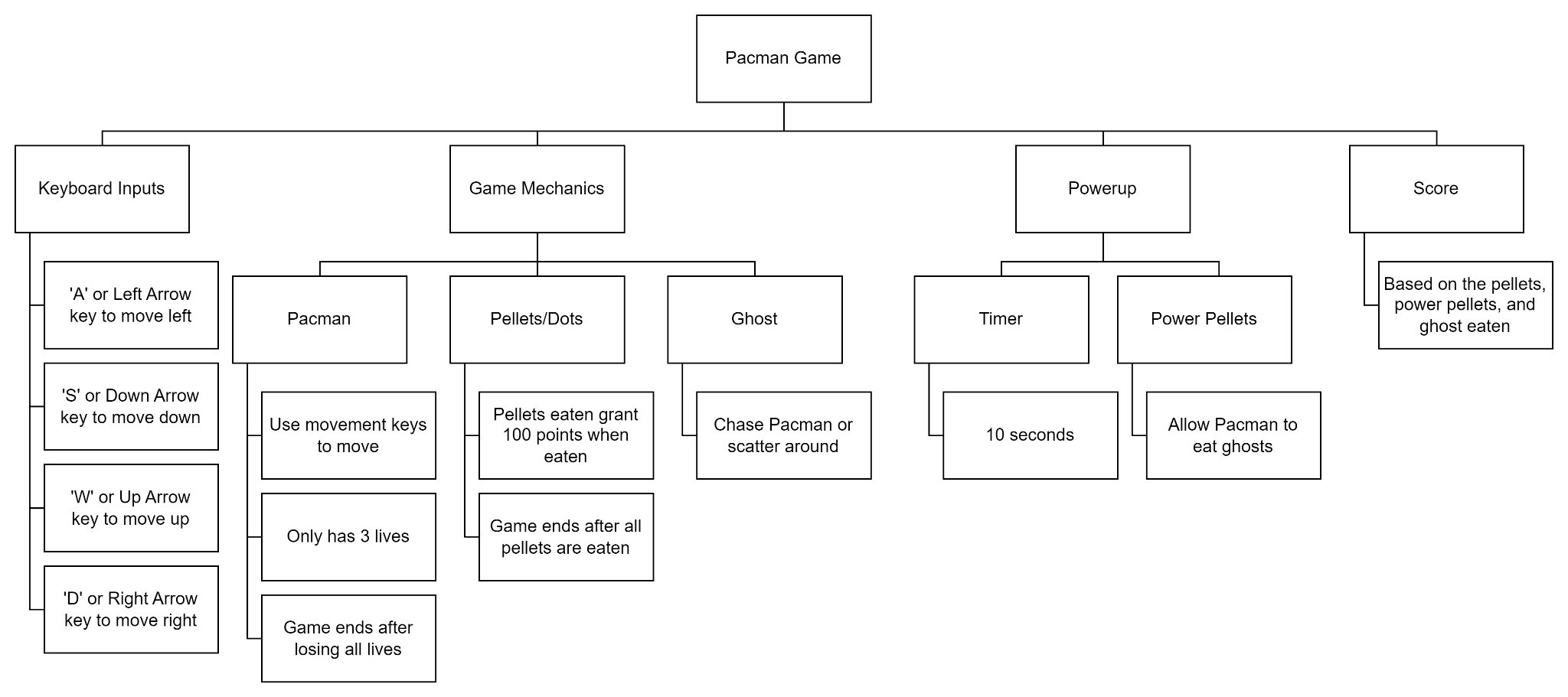
The character movement processes make sure that the characters, Pac-Man and the ghosts, are able to move inside the game. In this process, Pac-Man follows the direction the player is controlling it to, while the ghosts have two algorithms: scatter randomly or chase Pac-man. In the former, the ghosts 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.

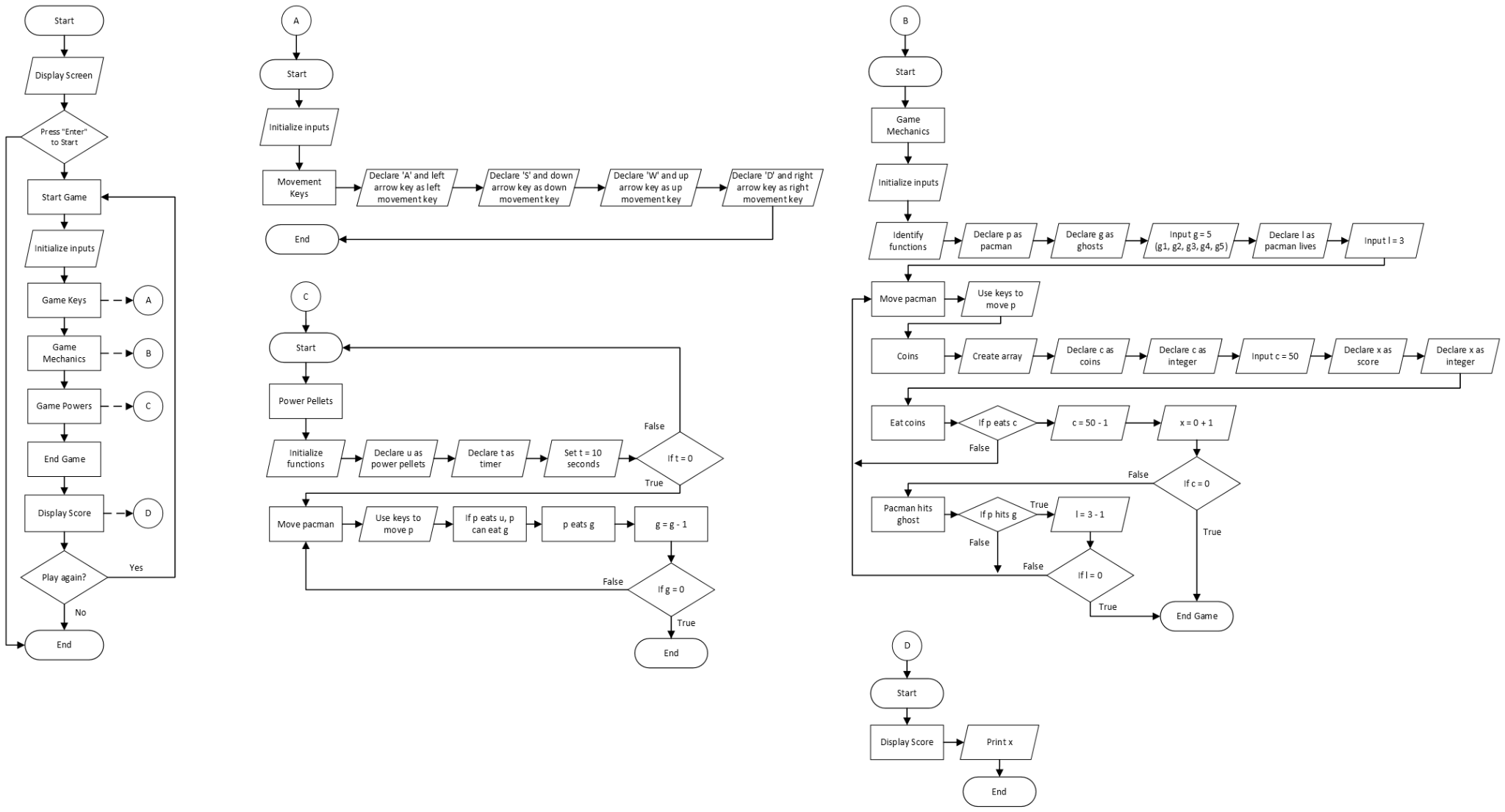
All these inputs and processes produce the output — the working replication of the Pac-Man game, with a twist of “multiverse madness.”

## Hierarchy Chart

Figure 2: Hierarchy of Modules/Processes

## 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*

## 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 | xxxx |  |  |  |  |
| **Game Mechanics Development** | Tabanao, Tiu |  |  |  |  | xxxx | xxxx | xxxx |  |  |
| **Game Coding** | Tabanao, Tiu |  |  |  |  |  | xxxx | xxxx | xxxx |  |
| **Game Design** | Tabanao, Tiu |  |  |  |  |  | xxxx | xxxx | xxxx |  |
| **Project/ Document Review** | Tabanao, Tiu |  |  |  |  |  |  |  | xxxx | xxxx |
| **Demonstration** | Tabanao, Tiu |  |  |  |  |  |  |  | xx | xxxx |

# Results

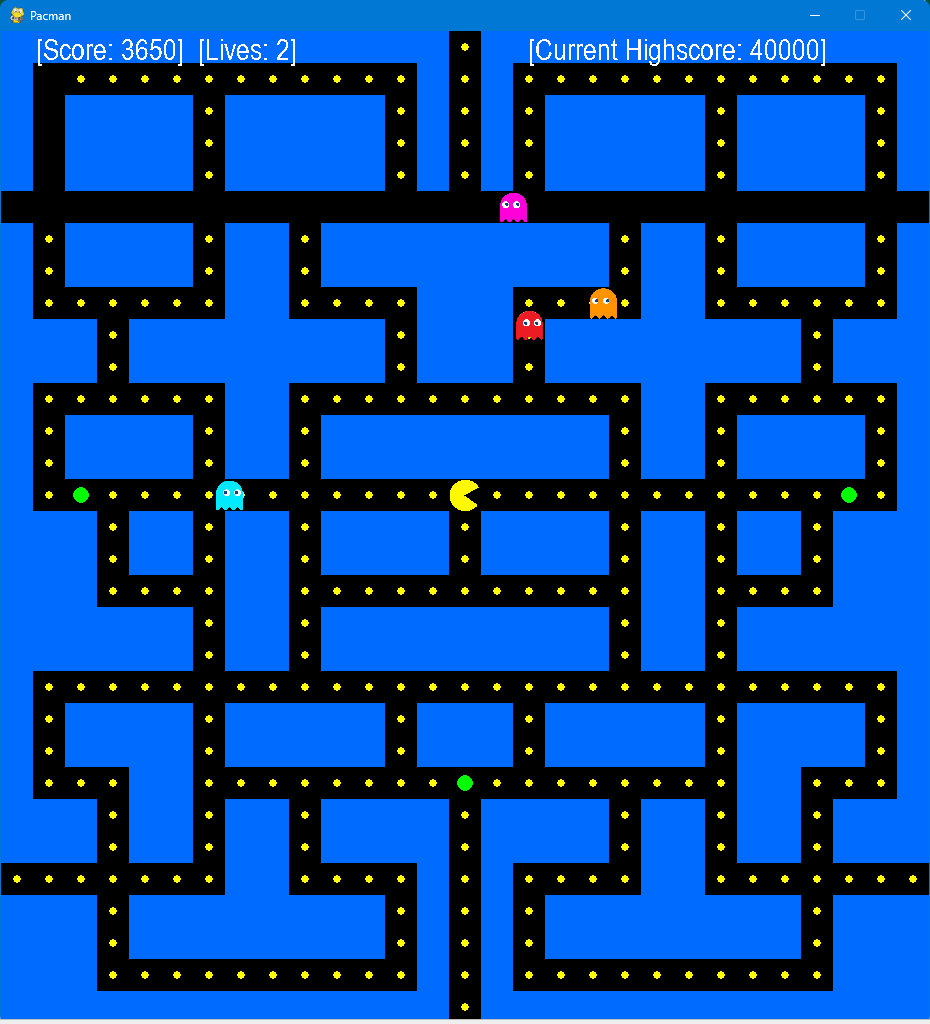
Figure 4: The Game with the Original Pac-Man Theme

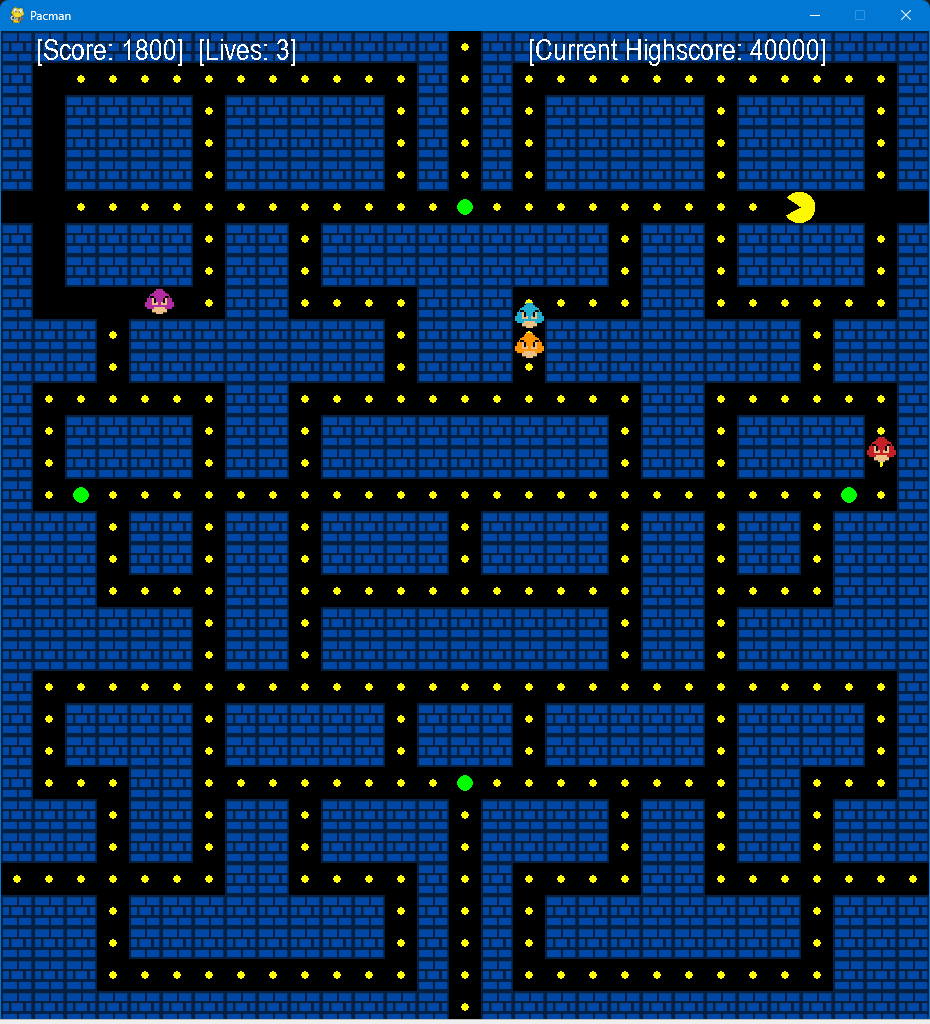
Figure 5: The Game with Super Mario Bros. Theme

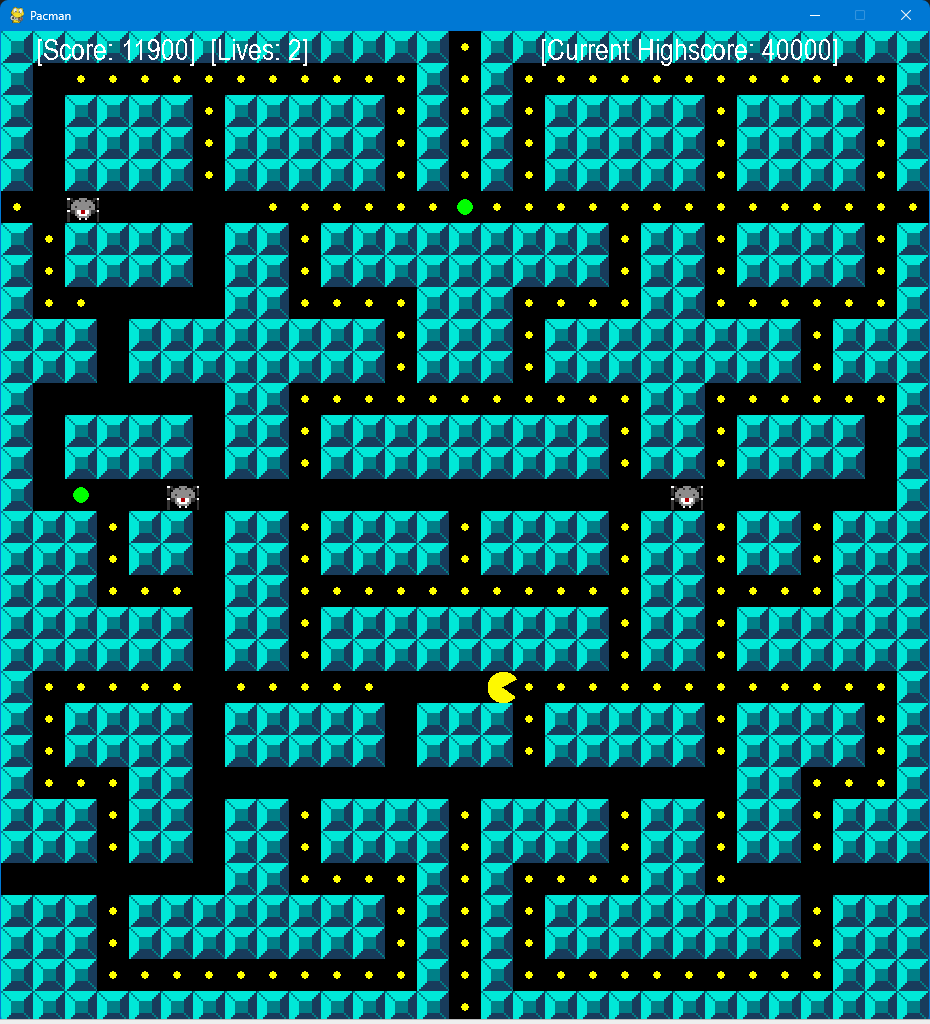
Figure 6: The Game with The Legend of Zelda Theme

Figure 7: The Game with Kirby Theme

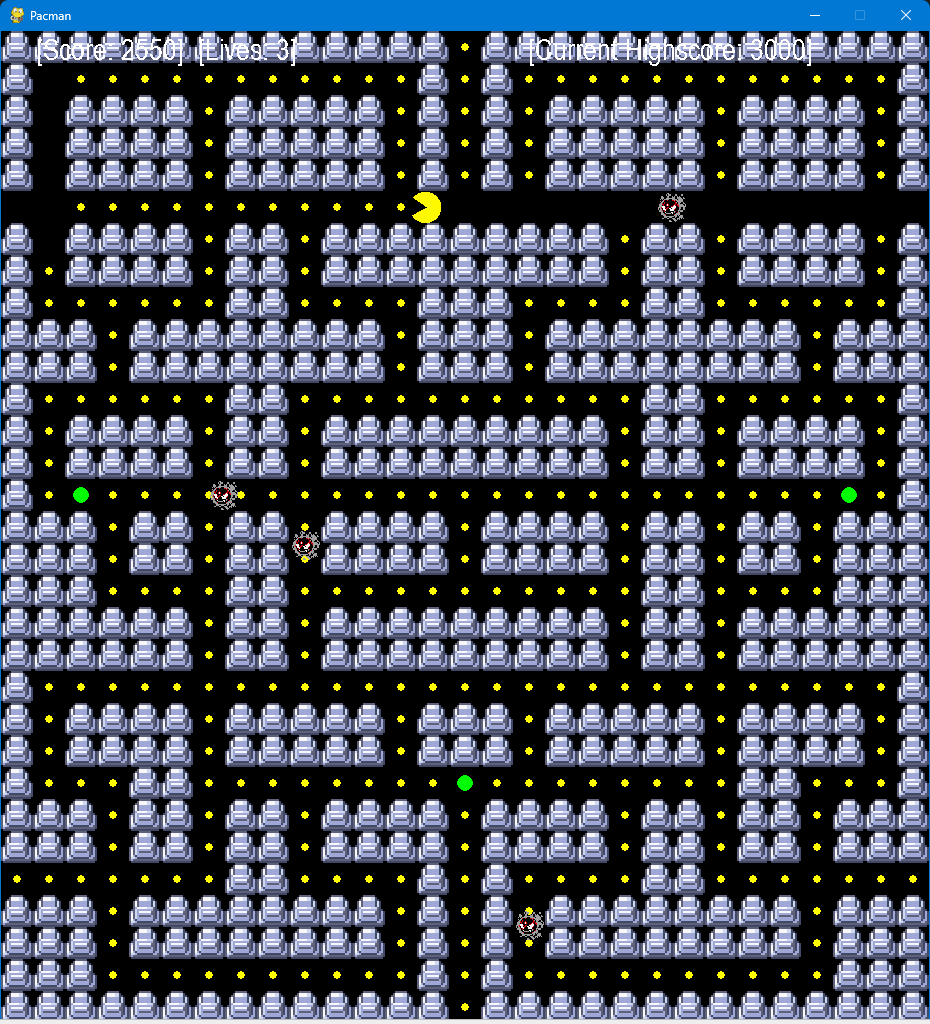
Figure 8: The Game with Pokemon Theme

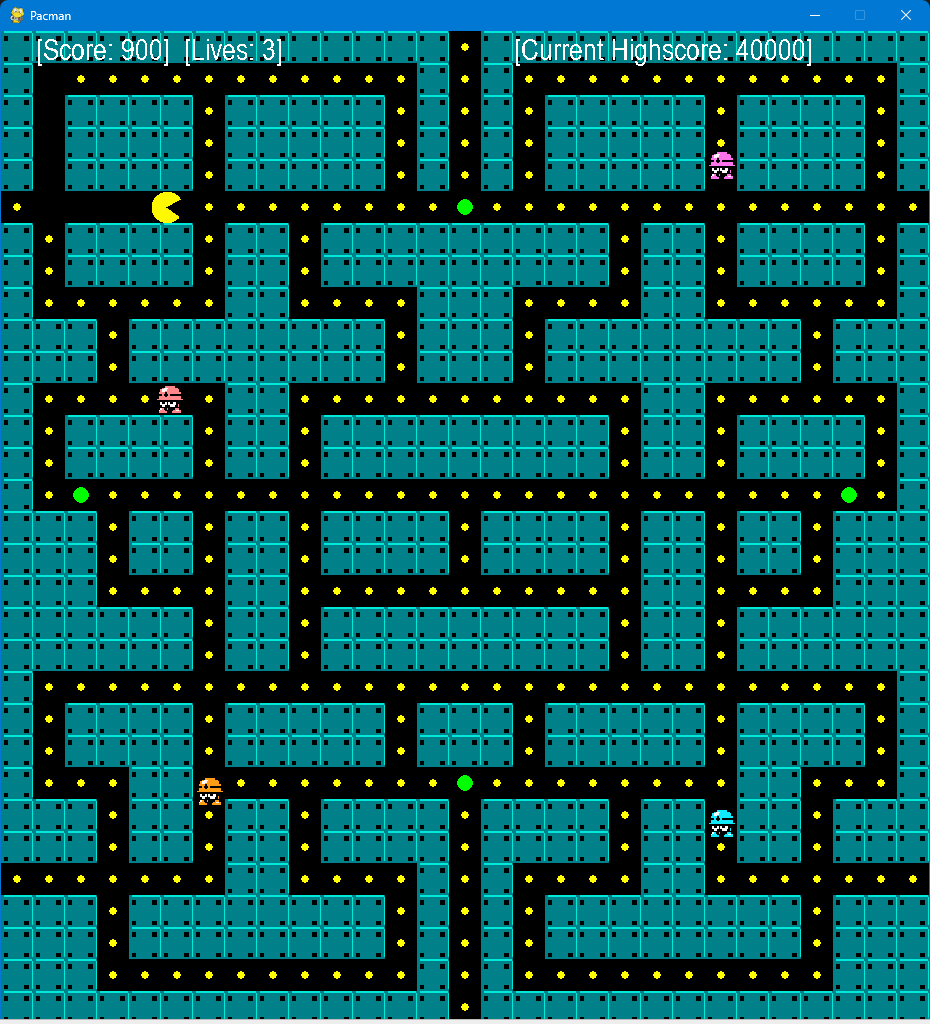
Figure 9: The Game with Mega Man Theme

Figure 10: The Game with Sonic the Hedgehog Theme

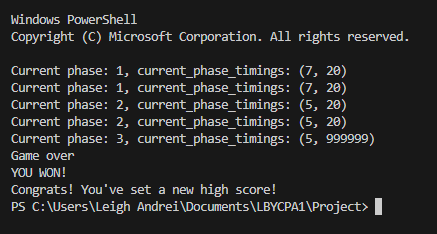
Figure 11.1: Behind the Scenes and Game Over Details (Player Wins with new Highest Score)

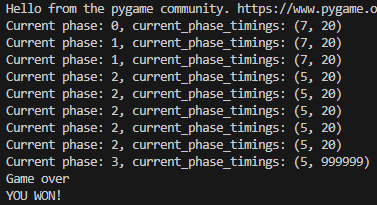
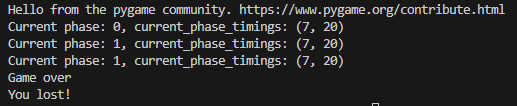
Figure 11.2: Behind the Scenes and Game Over Details (Player Wins)

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

# 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.

# 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](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](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

## 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](http://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 |
| --- |

c

| 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](https://code.visualstudio.com/) by the Microsoft team. Then, download the game files from the [GitHub repository](https://github.com/leightabanao/py3HadesProject). 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!

## 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",  "XXXXXXXXXXXXXX XXXXXXXXXXXXXX",  ]  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) |
| --- |

## 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% |

## 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 |