Assignment
Stage One Submission
2805ICT/3815ICT/7805ICT

Student name: Heang_SOK Student ID: s5204340 Enrolled Course Code: 7805ICT

Table of Contents

1.0 Project Planning and Documentation

1.1 Time Schedule

This section illustrates the expected time and actual time that this student needed to complete each task.

Task		Plan				Actual		
#	Task Name	Student	Planed Time	Cumulative Time	Finished Date	Time	Cumulative Time	Finished Date
1.1	Time Schedule	Heang Sok	1hr	1hr	09 Aug	30mn	30mn	28 Aug
1.2	Total Working Hours	Heang Sok	15mn	1hr15mn	09 Aug	5mn	35mn	28 Aug
1.3	Effort and Contribution Table	Heang Sok	15mn	1hr30mn	09 Aug	5mn	40mn	28 Aug
1.4	Version Control System	Heang Sok	3hr30mn	5hr	09 Aug	1hr	1hr40mn	09 Aug
2.1	Functional Requirements	Heang Sok	2hr	7hr	10 Aug	5hr	6hr40mn	23 Aug
2.2	Non-Functional Requirements	Heang Sok	2hr	9hr	10 Aug	5hr	11hr40mn	23 Aug
2.3	Use Case Diagram	Heang Sok	2hr	11hr	10 Aug	3hr	14hr40mn	23 Aug
2.4	Full Use Case Description	Heang Sok	2hr	13hr	11 Aug	3hr	17hr40mn	24 Aug
2.5	Requirement – Use Case Traceability Matrix	Heang Sok	2hr	15hr	11 Aug	3hr	20hr40mn	24 Aug
3.1	Class Diagram	Heang Sok	2hr	17hr	12 Aug	3hr	23hr40mn	26 Aug
3.2	Sequence Diagram	Heang Sok	2hr	19hr	12 Aug	4hr	23hr40mn	26 Aug
3.3	Activity Diagram	Heang Sok	2hr	21hr	12 Aug	4hr	23hr40mn	26 Aug
3.4	C & C View	Heang Sok	2hr	23hr	13 Aug	2hr	25hr40mn	26 Aug
3.5	Implementation Style View	Heang Sok	2hr	25hr	13 Aug	2hr	27hr40mn	26 Aug
3.6	Deployment View	Heang Sok	2hr	27hr	13 Aug	2hr	29hr40mn	26 Aug
4.0	Make Game	Heang Sok	50hr30mn	77hr30mn	18 Aug	157hr	186hr40mn	28 Aug
5.0	Video Link	Heang Sok	30mn	78hr	19 Aug	10mn	186hr50mn	28 Aug

Table 1: Time Schedule

1.2 Total Working Hours

Student Name (#ID)	Plan (hours)	Actual (hours)
HeangSOK_s5204340	76hr	184hr50mn
Total working hours	76hr	184hr50mn
Average working hours per person	76hr	184hr50mn

Table 2: Total Working Hours

1.3 Effort and Contribution Table

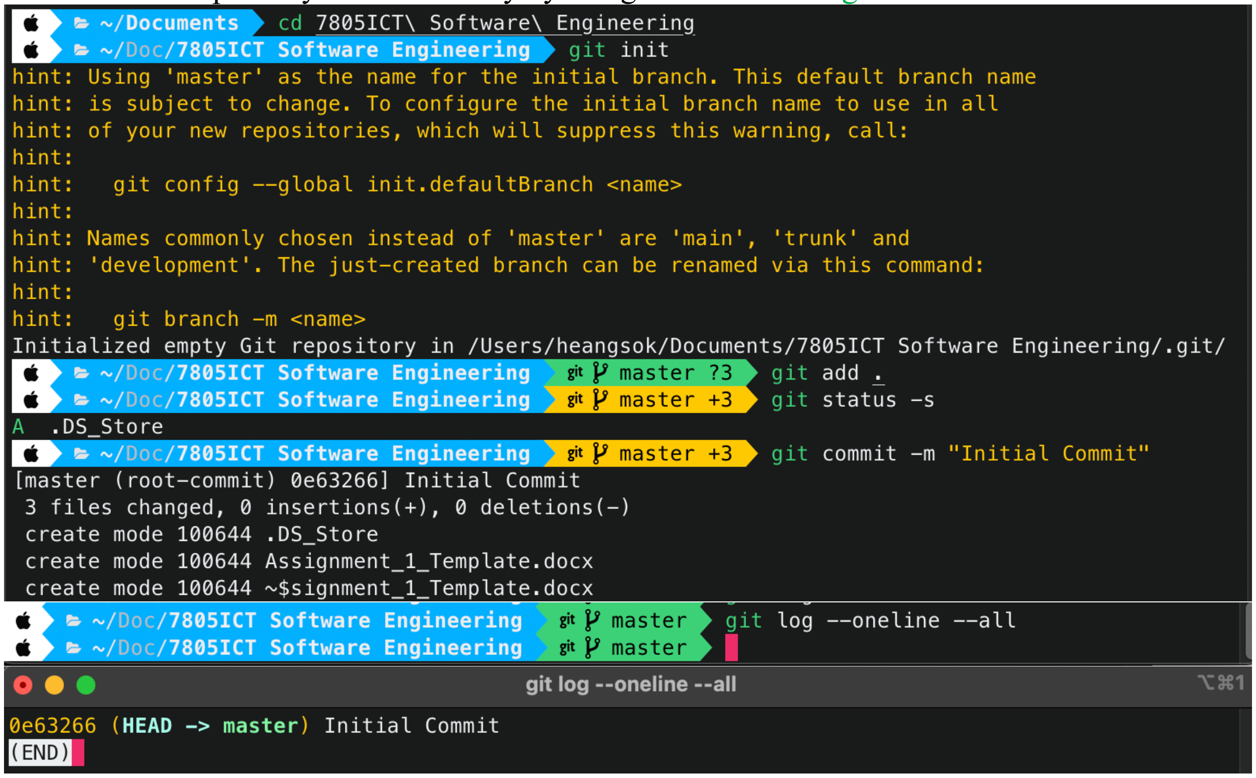
Student	**Effort Level*** (Rating from 0 – 5, the information is filled by the group)	**Contribution Level*** (Rating from 0 – 5, the information is filled by the group)	**Justification** If a student received level rating of 3 or less, your group need to give explanation for the low level rating
Heang Sok	5	5	
Total	5	5	

Table 3: Effort and Contribution Table

- *Level ratings, 5 = excellent, 4 = good, 3 = reasonable, 2 = poor, 1 = unacceptable, 0 = none

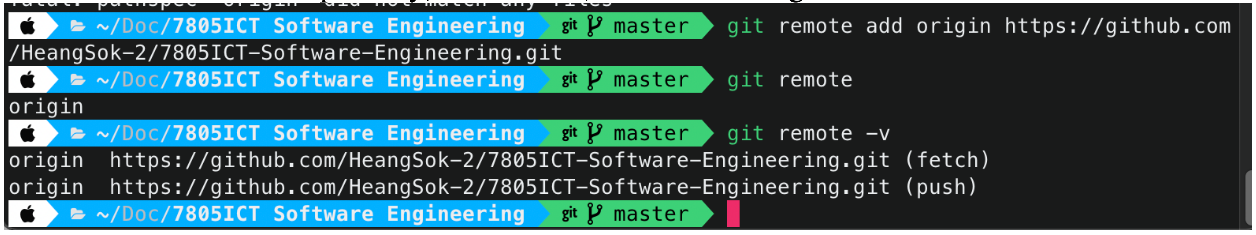
1.4 Version Control System

Create a local repository to work locally by using this command: git init.



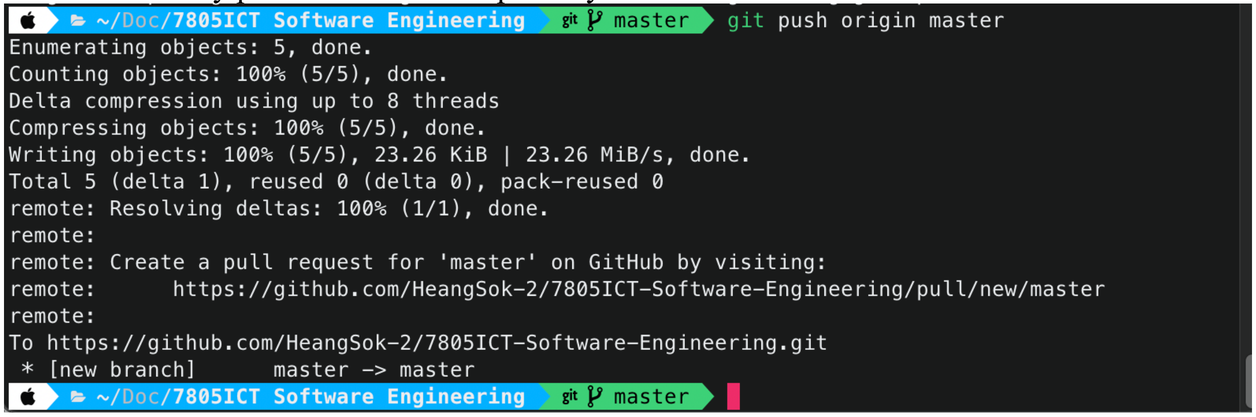
```
~/Documents  cd 7805ICT\ Software\ Engineering
~/Doc/7805ICT Software Engineering  git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/heangsok/Documents/7805ICT Software Engineering/.git/
~/Doc/7805ICT Software Engineering  git  master ?3  git add .
~/Doc/7805ICT Software Engineering  git  master +3  git status -s
A  .DS_Store
~/Doc/7805ICT Software Engineering  git  master +3  git commit -m "Initial Commit"
[master (root-commit) 0e63266] Initial Commit
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .DS_Store
 create mode 100644 Assignment_1_Template.docx
 create mode 100644 ~$signment_1_Template.docx
~/Doc/7805ICT Software Engineering  git  master  git log --oneline --all
~/Doc/7805ICT Software Engineering  git  master

git log --oneline --all                                        ⌥⌘1

0e63266 (HEAD -> master) Initial Commit
(END)
```

Connect to the remote repository; in this case, GitHub is being used.



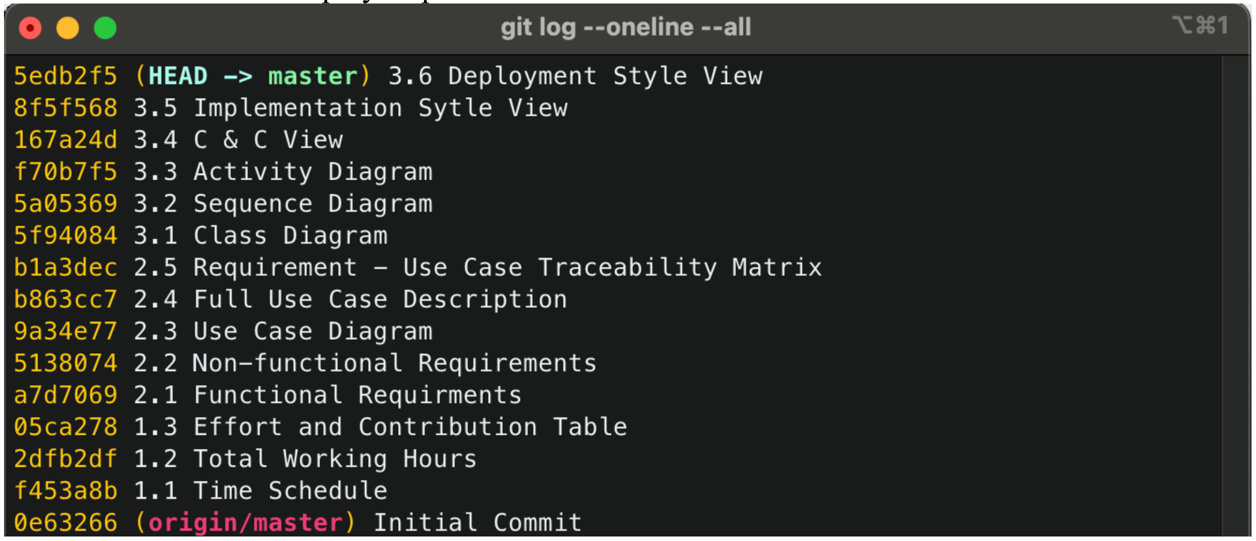
```
~/Doc/7805ICT Software Engineering  git  master  git remote add origin https://github.com
/HeangSok-2/7805ICT-Software-Engineering.git
~/Doc/7805ICT Software Engineering  git  master  git remote
origin
~/Doc/7805ICT Software Engineering  git  master  git remote -v
origin  https://github.com/HeangSok-2/7805ICT-Software-Engineering.git (fetch)
origin  https://github.com/HeangSok-2/7805ICT-Software-Engineering.git (push)
~/Doc/7805ICT Software Engineering  git  master
```

Push local repository to remote repository. As shown in the picture below, the initial commit has been successfully pushed to GitHub repository.



```
 ~/Doc/7805ICT Software Engineering  git  master  git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 23.26 KiB | 23.26 MiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/HeangSok-2/7805ICT-Software-Engineering/pull/new/master
remote:
To https://github.com/HeangSok-2/7805ICT-Software-Engineering.git
 * [new branch]      master -> master
 ~/Doc/7805ICT Software Engineering  git  master
```

The file is committed step by step.



```
git log --oneline --all
5edb2f5 (HEAD -> master) 3.6 Deployment Style View
8f5f568 3.5 Implementation Sytle View
167a24d 3.4 C & C View
f70b7f5 3.3 Activity Diagram
5a05369 3.2 Sequence Diagram
5f94084 3.1 Class Diagram
b1a3dec 2.5 Requirement - Use Case Traceability Matrix
b863cc7 2.4 Full Use Case Description
9a34e77 2.3 Use Case Diagram
5138074 2.2 Non-functional Requirements
a7d7069 2.1 Functional Requirments
05ca278 1.3 Effort and Contribution Table
2dfb2df 1.2 Total Working Hours
f453a8b 1.1 Time Schedule
0e63266 (origin/master) Initial Commit
```

2.0 Requirements Analysis

This section describes the Pac-Man system in terms of functional and non-functional requirements, use case diagram, full use case description, and use case traceability matrix by providing a brief overview of the subsystem and the reasons for its development.

2.1 Functional requirements

Pac-Man system is formed by multiple subsystems which comprise all the activities that support the functional requirements of the Pac-Man system. These requirements state what Pac-Man system must do and, how it must behave or react to runtime stimuli. All functional requirements are identified and described properly in table 4. Note: the functional requirement id starts with letter R.

ID	Subsystem	Functional Requirement	Description
R1	Start-up page subsystem	R1.1. Display title and logo of Pac-Man R1.2. Display year and course code R1.3. Display producer name R1.4. Display a play button, exit button, and configure button	• When users run the Pac-Man, this subsystem will be executed, and the start-up page will be displayed • the start-up page shall provide useful information to the users, such as the title and logo of Pac-Man; it also tells who the producer and his course code is • It allows users to play the game by clicking the play button, exit the game by clicking the exit button, and select a random/original map by clicking configure button
R2	Maze subsystem	R2.1. Define the locations of ghosts' home, walls, paths, Pac-Dots, Power Pellets, and Fruit R2.2. Define the start-up location of Pac-Man and Ghosts R2.3. The maze can be randomly generated	• Pac-Man's maze is arranged in a 2D array (36 rows x 28 columns) and can be randomly generated • Pac-Man and Ghosts can move through the predefined maze generated by this subsystem • Pac-Man and Ghosts appear at a certain position on the maze when start-up the game • Walls, paths, Pac-Dots, Power Pellets, and Fruit will be rendered from this subsystem
R3	Sound effect subsystem	R3.1. Allow the game to play different sound effects according to each state, such as starting, chasing, eating Pac-Dot, eating Power-Pellet, eating Fruit, eating Vulnerable Ghost, and death.	• This subsystem will automatically play a specific type of sound effect regarding the state of the game, such as starting, chasing, eating Pac-Dot, eating Power-Pellet, eating Fruit, eating Vulnerable Ghost, and death.
R4	Ghost subsystem (Also read Appendix A)	R4.1. Ghosts shall have 3 exclusive modes: Chase mode, Scatter mode, and Frightened Mode R4.2. Automatically and independently move	• There are four ghosts in Pac-Man, such as: -Blinky (red ghost): doggedly pursues Pac-Man -Pinky (pink ghost): move parallel to Pac-Man -Inky (cyan ghost): try to get Pac-Man in between Blinky and himself -Clyde (orange ghost): pursue Pac-Man only when far from him • Ghosts' Mode and Movement explained: ghosts enter Frightened mode only when Pac-Man eats one of the four energizers. At this point, all ghosts will turn into blue

			colour and avoid Pac-Man. In Scatter mode, ghosts give up the chase for a few seconds and head for their home corners. All ghosts will find their way to hunt and capture the Pac-Man in Chase mode
R5	Score Game subsystem (Also read Appendix A)	R5.1. Count scores each time Pac-Man eats Pac-Dot, Power Pellet, Fruit, and Vulnerable Ghost (Blue Ghost) R5.2. Save the highest score	• Whenever Pac-Man eats Pac-Dot, Power Pellet, Fruit, and Vulnerable Ghost the system starts counting and then compare the new score with the previous highest score. If the new score is higher than the previous highest score, the system overrides it with the new score • Scores are counted as follow if Pac-Man collides with Pac-Dot, Power Pellet, Fruit, and Vulnerable Ghost: ○ Pac-Dot: 10 points ○ Power Pellets: 50 points 🍒 Cherry: 100 points. 🍓 Strawberry: 300 points 🍊 Orange: 500 points 🍎 Apple: 700 points 🍈 Melon: 1000 points 👾 Galaxian Boss: 2000 points 🔔 Bell: 3000 points 🔑 Key: 5000 points 👻First Collison with a vulnerable ghost: 200 points 👻Second Collison with a vulnerable ghost: 400 points 👻Third Collison with a vulnerable ghost: 800 points
R6	Pac-Man subsystem	R6.1. Pac-Man dies when the ghosts capture him R6.2. Pac-Man can be moved forward, turned left, turned right, and turned back by console (keyboard) R6.3. Pac-Man has 3 lives	• Pac-Man has 3 lives; whenever the ghosts capture the Pac-Man, his lives are reduced by 1. The game will be over when there is no life left. • Pac-Man can be controlled by a keyboard to move forward, turn left, turn right, and turn back.
R7	Render Subsystem	R7.1. Its function is to render all the graphic interfaces, such as Text, Pac-Dot, Power Pellet, Pac-Man, Ghosts, Walls, and Buttons.	• This subsystem will render all the graphic interfaces on the screen and allow users to interact with the Pac-Man System.
R8	Level-up Subsystem	R8.1. Not required in this project (also read stage two of the assignment)	• Not required in this project (also read stage two of the assignment)

Table 4: Functional Requirements

2.2 Non-functional requirements

Non-functional requirements or quality attributes also have a huge influence on the whole Pac-Man system because it enhances values to user experience and ensures the reliability of the Pac-Man system at runtime. Put it simply, it focuses on the characteristics of the system other than functions to be performed. To distinguish non-functional from the functional requirements, this paper uses the FURPS+ framework, which includes usability, reliability, performance, security, design constrain, and implementation. All non-functional requirements are identified and listed in table 5.

Non-Functional Requirements	
FURPS+ Category	Requirement Description
Usability	• This Pac-Man system can be run seamlessly on Window and Mac Operating System • This Pac-Man system can be played on a local machine (PC and Laptop); it does not require internet • Support any brands of keyboard devices for controlling Pac-Man; this includes Window's keyboard and Apple's magic keyboard • The user interface is designed with big icons and images; thus, it gives a friendly environment and reduces stress on users' eyes when playing • The system's start-up menu is simple and easy to execute with only a single click of a button
Reliability	• The system can run without delay time • The system does not slow down the computer machines • Users can play the game anytime on their computer devices (PC and Laptop) • The system does not cause delay time and crush other systems • Users' highest score will be archived safely and prevented from lost • The sound effects (extension is .wav) are supported by all platforms
Performance	• The system consumes low memory (about 16kb of RAM) to execute since the algorithm behind the Pac-Man system uses only 8 bits (or 255) to build the game • The sound effect volume is played based on the operating system volume • The system support one player at a time
Security	• Since the original Pac-Man was built on 8 bits, users cannot go over level 255, or the screen will split and display a weird behaviour (also read appendix A). This weird behaviour sometimes may slow down the computer or cause the computer to reboot which is prone to the loss of unsaved files. To prevent this incident, in this project, we do not implement the level-up subsystem (also see Table 4). This means that when users pass level 1 the game will pop up a message saying that the users win the game • Since the system is run locally, it is free from any kind of internet attack
Design Constraints	• Standard computer machine requirements for this project: ○ Processor: 1 gigahertz (GHz) or faster processor ○ RAM: 4GB ○ Storage: 32GB • This system requires the following working environment to run: ○ Operating system: Window 10, Mac (Big Sur)
Implementation	• This system is implemented by: ○ Programming Language: **Python version 3+** ○ Sound effects: **wav format** ○ Images and icons: **png format** ○ High score file: **txt format** ○ Tools: ▪ **PyCharm** version 2021.2 is used to run Python Language ▪ **Text editor** is used to edit text ▪ **Sony Vegas** pro 365 is used to edit sound effect ▪ **Adobe Photoshop** is used to edit images and icons for user interface (UI)

Table 5: Non-Functional Requirements

2.3 Use Case Diagram

A use case diagram is being used to demonstrate Pac-Man system when interacts with a player. As shown in figure1, the player can start the game by clicking the Play Button. After the game is started the system renders window, move ghosts, and play the sound effects. It is optional for the player to configure the maze before starting the game. To exit the game, the player just needs to click the Exit Button, and the player can move the Pac-Man by controlling a keyboard.

On the other hand, to score the game, the player needs to eat Pac-Dots, Power-Pellets, Fruit, or Vulnerable Ghosts. The score will be returned as follow:

- Return 10 points if the Pac-Man eats Pac-Dot
- Return 50 points if the Pac-Man eats Power-Pellet
- Return from 100 to 5000 points if the Pac-Man eats Fruit (also see Appendix A)
- Return from 200 to 5000 points if the Pac-Man eats Vulnerable Ghost (also see Appendix A)

The system will automatically compare the new score with the previous highest score at the end of the game; if the new score is higher than the previous highest score, it will override the previous highest score with the new score. Lastly, the player can move the Pac-Man through maze.

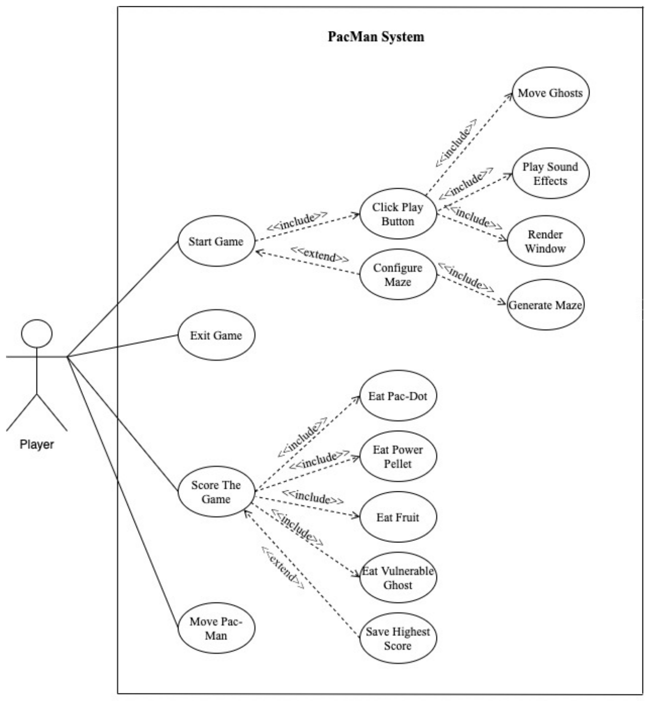


Figure 1: Use Case Diagram for Pac-Man System

2.4 Full use case description

This section will illustrate thoroughly about Score Game use case which belongs to Score Game Subsystem. In table 6 below, to score the game, the actor (player) can either run the game directly or configure the maze first then run the game. Next, the actor needs to move the Pac-

Man to collide with Pac-Dot, Power Pellet, Fruit, or Vulnerable Ghost to score the game. Note that the Pac-Man will not receive any points from colliding with normal ghosts; he will lose his life instead. After the actor runs out of lives, the game is over, and the system starts to compare the new score with the previous highest score. If the new score is higher than the previous highest score, the system overrides the previous highest score with the new score.

Fully Developed Use Case Description		
Use Case Name:	Score Game	
Scenario:	The player wants to win and score the game	
Triggering Event:	When Pac-Man collides (eats) with Pac-Dot, Power Pellet, Fruit, or Vulnerable Ghost (blue ghost), the system will count the score	
Brief Description:	During the game, Pac-Man can move through the maze and eat Pac-Dot, Power Pellet, Fruit, or Vulnerable Ghost (blue ghost) in order to score	
Users/Actor:	Player	
Related Use Cases:	Star Game use case, Move Pac-Man use case, Render Window use case, Play Sound Effect use case, Move Ghosts use case	
Preconditions:	• The player runs the game • The player clicks on a play button, or • The player configures the maze first and then clicks on a play button • The player moves Pac-Man to collide with Pac-Dot, Power Pellet, Fruit, or Vulnerable Ghost	
Postconditions:	• After eating all the Pac-Dots and Power Pellets, the player wins the game • After the player runs out of life, the game is over. • The system compares the new score with the previous highest score; if the new score is higher than the previous highest score, the system overrides the previous highest score with the new score	
Flow of Activities	Player	System
	1. When the player moves Pac-Man to collide with Pac-Dot	1.1. The system returns and increases 10 points to the score system
	2. When the player moves Pac-Man to collide with Power Pellet	2.1. The system returns and increases 50 points to the score system
	3. When the player moves Pac-Man to collide with Fruit	3.1. The system returns and increases points as follow: 🍒 Cherry: 100 points. 🍓 Strawberry: 300 points 🍊 Orange: 500 points 🍎 Apple: 700 points 🍈 Melon: 1000 points 👾 Galaxian Boss: 2000 points 🔔 Bell: 3000 points 🔑 Key: 5000 points
	4. When the player moves Pac-Man to collide with Vulnerable Ghost	4.1. The system returns and increases points as follow: 👻First Collison - 200 points 👻Second Collison - 400 points 👻Third Collison - 800 points 👻Fourth Collison - 1600 points
Exception Conditions	4. The system does not return and increase points when Pac-Man collides with a normal ghost (Pac-man lives will be decreased instead)	

Table 6: Full Developed Use Case Description

2.5 Requirement - use case traceability matrix

This project utilises the Use Case Traceability Matrix to map and show the relationship between the functional requirements with the use cases. It ensures that all the subsystems of the Pac-Man system are checked and matched with the player's requirements before the deployment of this system. Table 7 shows that all the use cases are tested and worked well with the proposed requests below:

Use cases:

UC1: Start Game
UC2: Exit Game
UC3: Score the Game
UC4: Move Pac-Man
UC5: Configure Maze
UC6: Return 10 Points
UC7: Return 50 Points
UC8: Return From 100 to 5000 Points
UC9: Return From 200 to 1600 Points
UC10: Save Highest Score
UC11: Move Ghosts
UC12: Play Sound Effects
UC13: Render Window

Requests:

REQ1: Player requests to start the game.

Explanation:

- When the player requests to start the game, the system will render the window (U13), play sound effects (UC12), and move ghosts (UC11).

REQ2: Player configure the maze first, then start the game

Explanation:

- The player can also configure the maze (UC5) before he/she requests to start the game (this part is optional for the player).

REQ3: Player moves Pac-Man to collide with Pac-Dot and scores

Explanation:

- If the player moves the Pac-Man (UC4) to collide with Pac-Dot, he/she will score the game (UC3) and the system will return 10 points (UC6); the Pac-Dot music will be played as well (UC12).

REQ4: Player moves Pac-Man to collide with Power Pellet and scores

Explanation:

- If the player decides to move the Pac-Man (UC4) to collide with the Power Pellets, he/she will score the game again (UC3), but the system will return 50 points (UC7) instead; the Power Pellet music will be played as well (UC12).

REQ5: Player moves Pac-Man to collide with Fruit and scores

Explanation:

- If the player decides to move the Pac-Man (UC4) to collide with the Fruit, he/she will score the game once again (UC3), but the system will return from 100 to 500 points depend on what type of fruit the Pac-Man collides with (also see appendix A); the Fruit music will be played as well (UC12).

REQ6: Player moves Pac-Man to collide with Vulnerable Ghosts (Blue Ghosts) and score

Explanation:

- If the player moves the Pac-Man (UC4) to collide with the Vulnerable Ghosts (Blue Ghosts), he/she will score the game (UC3), and the system will return from 200 to 1600 points (also see appendix A); the Vulnerable Ghost music will be played as well (UC12).

REQ7: Player exit the game

Explanation:

- If the player wants to close the game, use case UC2 will be used, and the system starts to compare the new score with the previous highest score. If the new score is higher than the previous highest score, the system overrides the previous highest score with the new score (UC10)

Request	**Player's Use Case**				**System's Use Case**								
	UC1	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**	**UC9**	**UC10**	**UC11**	**UC12**	**UC13**
REQ1	X										X	X	X
REQ2	X				X						X	X	X
REQ3			X	X		X						X	X
REQ4			X	X			X					X	X
REQ5			X	X				X				X	X
REQ6			X	X					X			X	X
REQ7		X								X			

Table 7: Requirement Traceability Matrix for Use Cases

3.0 Design and software architecture

3.1 Class diagram

Figure 2 shows the main building block of each class that is required to build the PacMan System. This diagram generates a general conceptual model of PacMan Systems structure which can be translated into programming language structure.

In terms of composition, the GameSystem class has one RenderWindow class and an Object class. These classes are connected using the black diamond at the end of each line because the RenderWindow and Object class cannot stand alone, and it depends heavily on the GameSystem class to run (also see figure 2).

In terms of generalisation or inheritance, the Object class has many children classes, such as PacMan, PacDot, PowerPellet, Fruit, Maze, and Ghost class. In addition, the Ghost class is also divided into four different classes, such as RedGhost, PinkGhost, OrangeGhost, and CyanGhost. Note that Object and Ghost class are also called abstract class that is why it is written in italic in the class diagram below (also see figure 2).

The relationships of this class diagram are listed as follow:

- GameSystem class can have only one RenderWindow class, but it can have one or more objects

- One or more ghosts can walk through one maze only, and one PacMan will walk through this maze
- One or more ghosts chase one PacMan, and one PacMan can eat one or more ghosts when they are in the frightened mode
- One PacMan can eat one or more Pac-Dot, Fruit, and Power Pellet
- One RenderWindow class can render one or more objects.

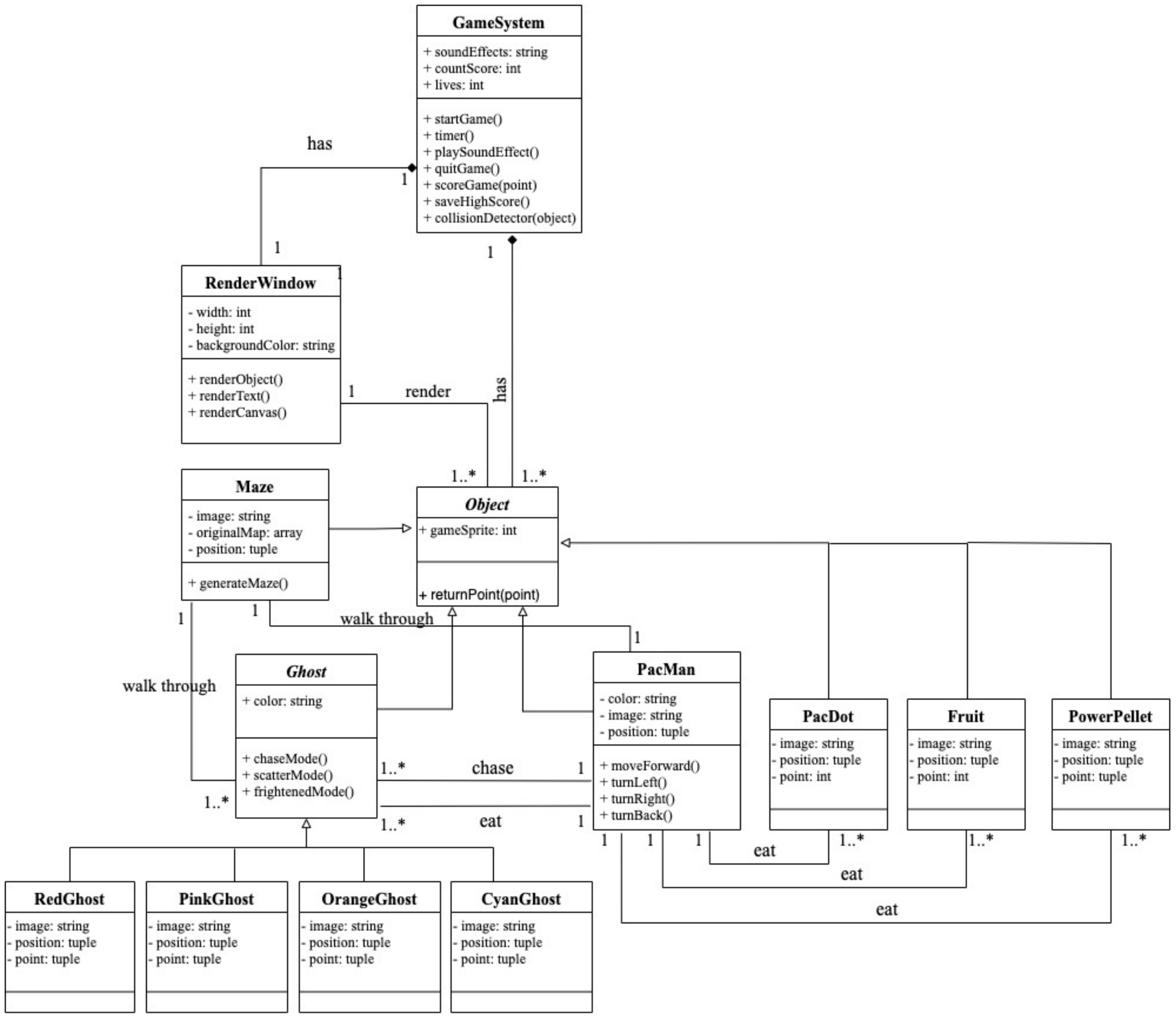


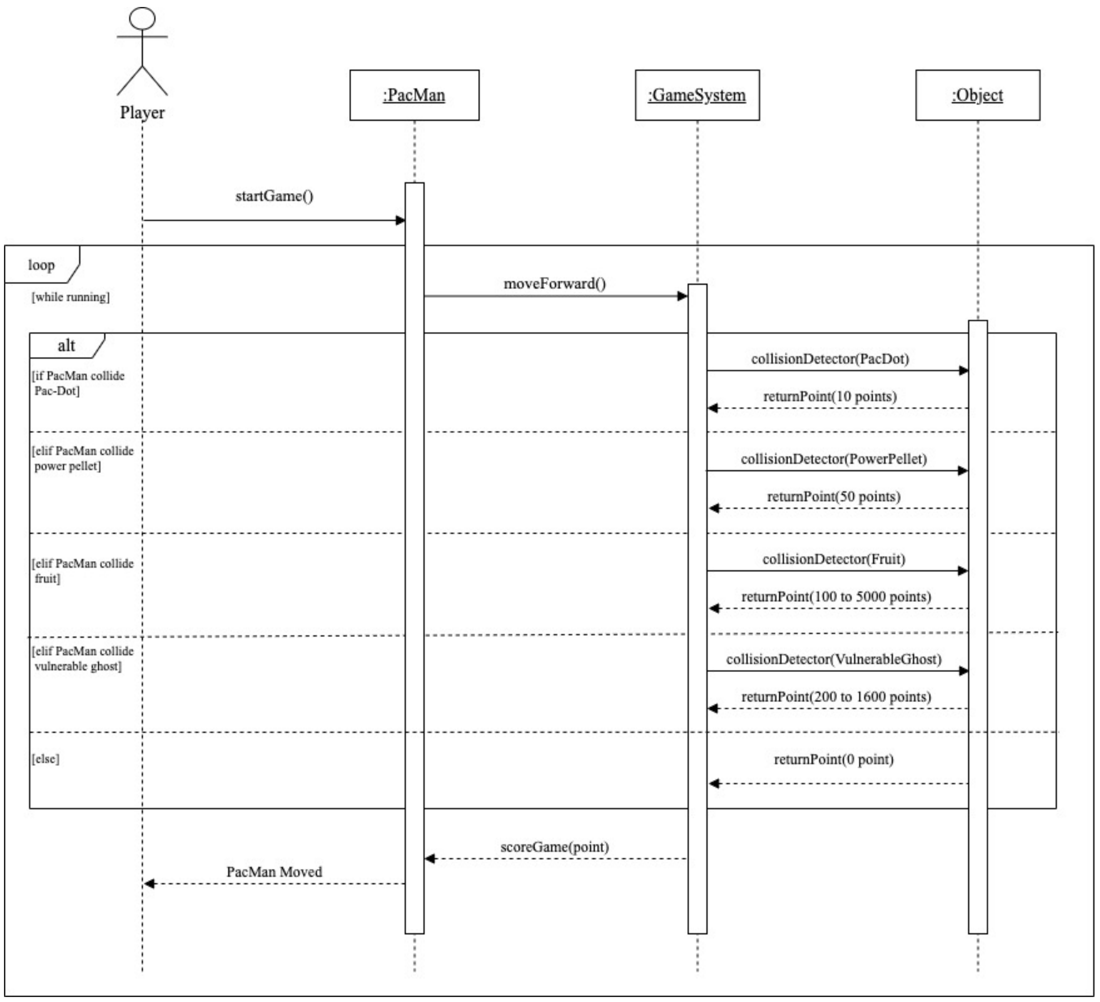
Figure 2: Class Diagram

3.2 Sequence diagram

Figure 3 illustrates the sequence of messages between the player and system. The scenario for this sequence diagram is when the player wants to score, he needs to move the Pac-Man first. In this case, the player sends a message to the PacMan class to move forward. Then the

GameSystem class checks if the PacMan collides with the Pac-Dot or not. If yes, the Object class returns 10 points. If not, the GameSystem class continues to check if the PacMan collides with the Power Pellet or not. If it collides with the Power Pellet, the Object class returns 50 points. If not, the GameSystem class continues to check if the PacMan collides with the Fruit or not. If it collides with the Fruit, the Object class returns from 100 to 5000 points depending on what types of Fruit that it collides with. If not, the GameSystem class continues to check if the PacMan collides with the Vulnerable Ghost or not. If it collides with the Vulnerable Ghost, the Object class returns from 200 to 1600 points. Lastly, the Object class will return zero point if the PacMan collides with nothing. The scoreGame(point) function will be executed after the points are sent from the object class.

This process loops through the entire period that the game is running. It stops when the game is over or there is no more Pac-Dot or Power Pellet.



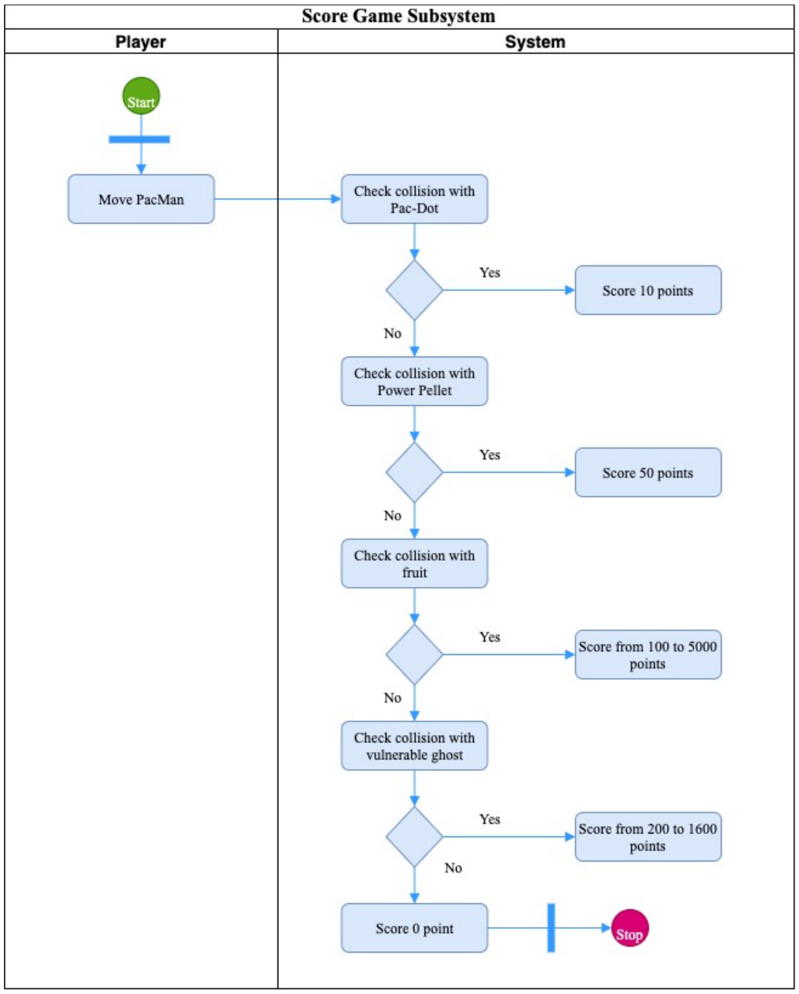
Player
:PacMan
:GameSystem
:Object
startGame()
loop
[while running]
moveForward()
alt
[if PacMan collide Pac-Dot]
collisionDetector(PacDot)
returnPoint(10 points)
[elif PacMan collide power pellet]
collisionDetector(PowerPellet)
returnPoint(50 points)
[elif PacMan collide fruit]
collisionDetector(Fruit)
returnPoint(100 to 5000 points)
[elif PacMan collide vulnerable ghost]
collisionDetector(VulnerableGhost)
returnPoint(200 to 1600 points)
[else]
returnPoint(0 point)
scoreGame(point)
PacMan Moved

Figure 3: Sequence Diagram for Scoring Game Scenario

3.3 Activity diagram

The bellow diagram elaborates all the activities that are required in Score Game Subsystem from a starting point to a finish point. Once again, to score the game, the player needs to move the Pac-Man across the maze. During the running time, the system keeps checking again and again for the collision of Pac-Man with other objects, such as Pac-Dot, Power Pellet, Fruit, or Vulnerable Ghost. Therefore, all the actions are placed between a pair of synchronization bars. The loop will end when the game is stopped. This activity diagram also tells the following decision behaviours of the system:

- If the Pac-Man collides with the Pac-Dot, the system scores 10 points. If not, the system checks if Pac-Man collides with the Power Pellet.
- If the Pac-Man collides with the Power Pellet, the system scores 50 points. If not, the system checks if Pac-Man collides with the Fruit.
- If the Pac-Man collides with the Fruit, the system scores from 100 to 5000 points (also read Appendix A). If not the system check if Pac-Man collides with the Vulnerable Ghost.
- If the Pac-Man collides with the Vulnerable Ghost, the system scores from 200 to 1000 points (also read Appendix A). if not, the system scores 0 point.



Score Game Subsystem
Player
System
Start
Move PacMan
Check collision with Pac-Dot
Yes
Score 10 points
No
Check collision with Power Pellet
Yes
Score 50 points
No
Check collision with fruit
Yes
Score from 100 to 5000 points
No
Check collision with vulnerable ghost
Yes
Score from 200 to 1600 points
No
Score 0 point
Stop

Figure 4: Activity Diagram for Score Game Subsystem

3.4 C & C View

The following component and connector view is built to help modelling implementation details and double-check if all the use cases of the system are covered by the plan or not. This student uses Model View Control (MVC) technique to cover the entire system.

On one hand, the Game Model component will be responsible for all the logic and algorithm (include AI) behind the system that the player interacts with. On the other hand, the Game Controller component processes all the player's requests; then manipulate the Game Model and interact with the user inface (Game View). Game View component is used to render all the graphic interface, such as images, texts, and buttons. Lastly, the Game HighestScore File component is used for storing data or information of the Game.

As shown in the figure 5, the player needs to invoke with the Game Controller first in other to update the user interface (Game view) and manipulate the Game Model. When the Game Model and Game View are updated it send a respond back to the Game Controller. For the Game HighestScore File component, it receives the data from the Game Model, and it sends the data back to Game Model to update the highest score on the screen.

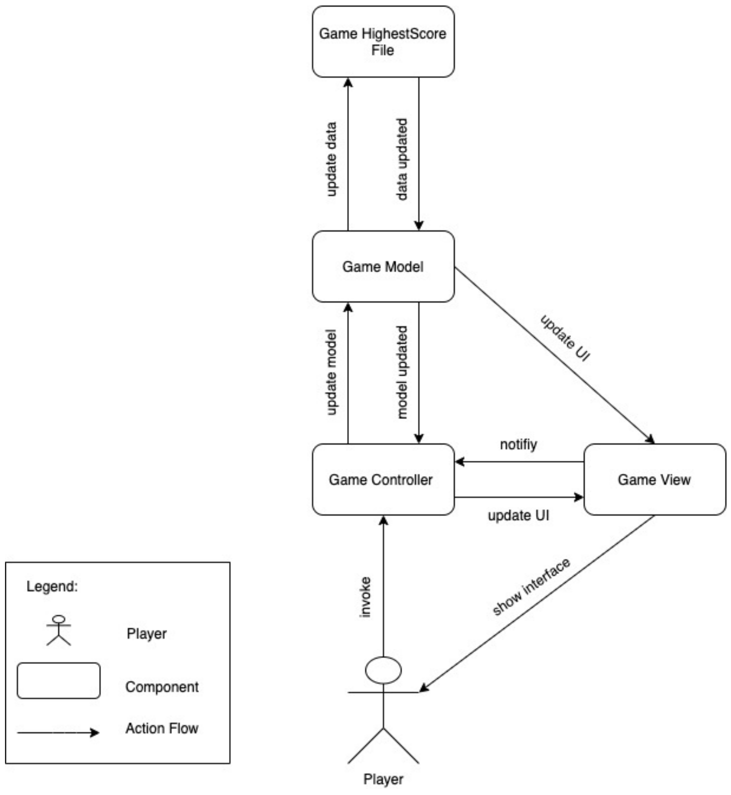
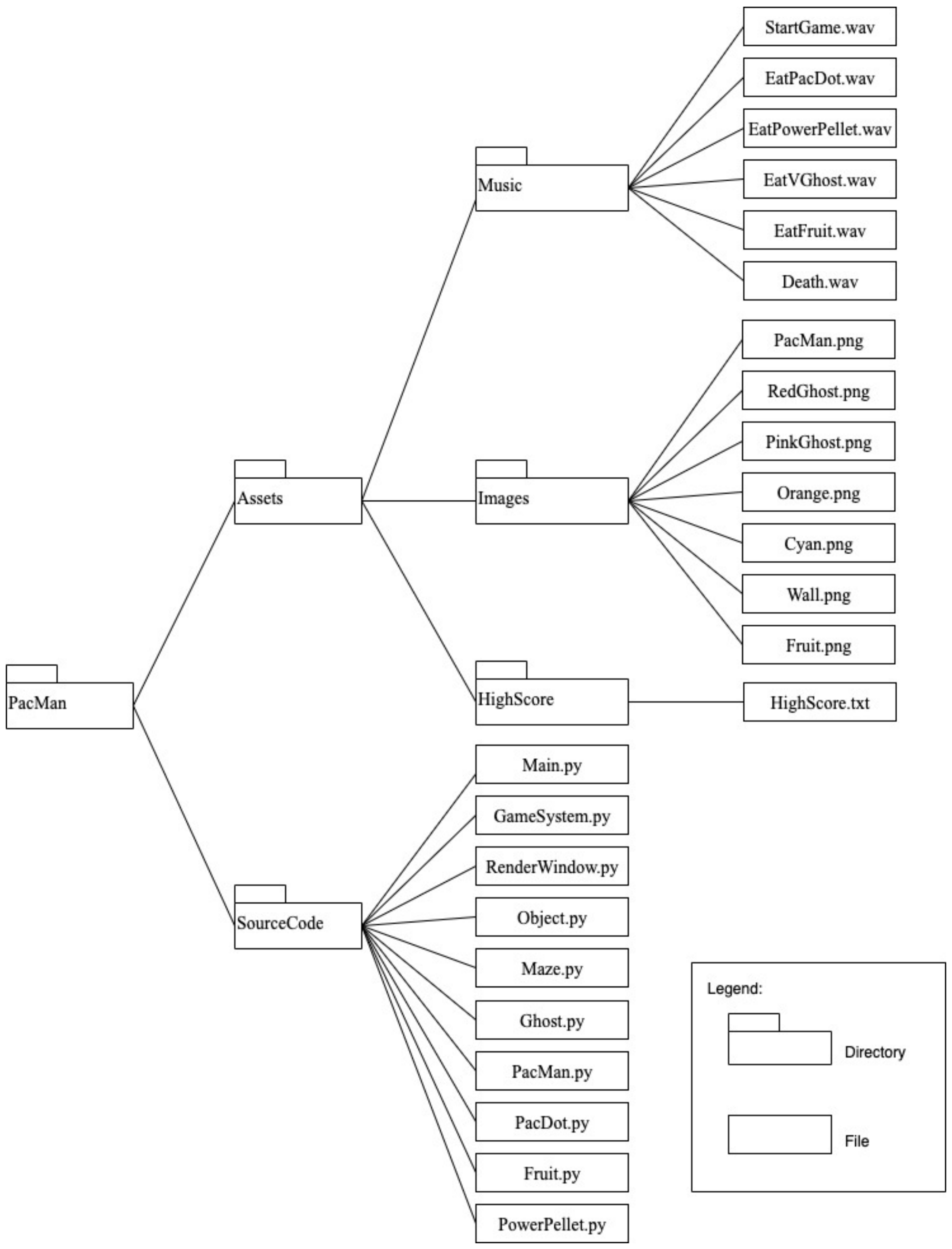


Figure 5: The Component and Connector View

3.5 Implementation style view

Figure 6: Implementation View for The Entire System



The purpose of the above figure is to capture the architectural decisions made for this project. It enhances the visibility of the system and how the system flows from the top to the less priority. In this project, all the directories and files will be constructed as shown in figure 6. There is a master folder called PacMan, and it will store all the components and source codes to build this game. By convention, a game developer usually stores all the components, such as music, images, and text file in a directory named Assets. For images, game developers usually create a directory to group them together; the same goes for music and text files.

On the other hand, all source code will be grouped in a directory named SorceCode. Table 8 below describes each module in detail.

No	Module	Description
1	Main.py	As a Pythonista, it is recommended that the developer executes the program in a module called main.py. In this module, there are only a few lines of code that call necessary functions from other modules.
2	GameSystem.py	This module is responsible for starting and quitting the game. Moreover, it also handles collision detection, music, and score.
3	RenderWindow.py	This module will be responsible for rendering windows, texts, and objects, such as Maze, PacMan, Pac-Dot, Power Pellet, Ghost, and Fruit.
4	Object.py	Since PacMan is a type of tile game, the sprite concept is being used in this module. This module stores the size of each sprite that is used to create Maze, PacMan, Pac-Dot, Power Pellet, Ghost, and Fruit. It also has a function to return scores when there is a collision between PacMan and Objects.
5	Maze.py	The main function of this module is to generate a maze from a predefine array or to randomly generate a maze using "Recursive Division" method.
6	Ghost.py	This module handles all the functions and attribute that associated with the ghost. The functions include chasing mode, frightened mode, and scatter mode. In terms of attribute, this module comprises colour, image, position, and point.
7	PacMan.py	This module will handle the movement of the PacMan, such as move forward, turn left, turn right, and turn back. The PacMan information, such as colour, image, and position, is also mentioned here.
8	PacDot.py	This module contains all the Pac-Dot information that is being used to render on the screen. This information includes image, position, and point.
9	Fruit.py	This module contains all the Power Pellet information that is being used to render on the screen. This information includes image, position, and point.
10	PowerPellet.py	This module contains all the Fruit information that is being used to render on the screen. This information includes image, position, and point.

Table 8: Source Code Description

3.6 Deployment style view

The deployment view is one of the five architectural view models in software architecture. It illustrates all the environments that the system needs to be deployed. In this case, figure 7, the PacMan game can run locally on a computer and across different platforms, such as Mac and Window. Since the game is written in the Python programming language, Python version 3 is needed to be installed in the computer so that the game system can be executed.

The deployment environment of this project is listed as follow:

Deployment Environment	Requirements
Device/Hardware:	Personal Computer/laptop
Operating System:	Window 10/ Mac Big Sur
Programming Language:	Python version 3
Game System:	PacMan Game

Table 9: Deployment Environment

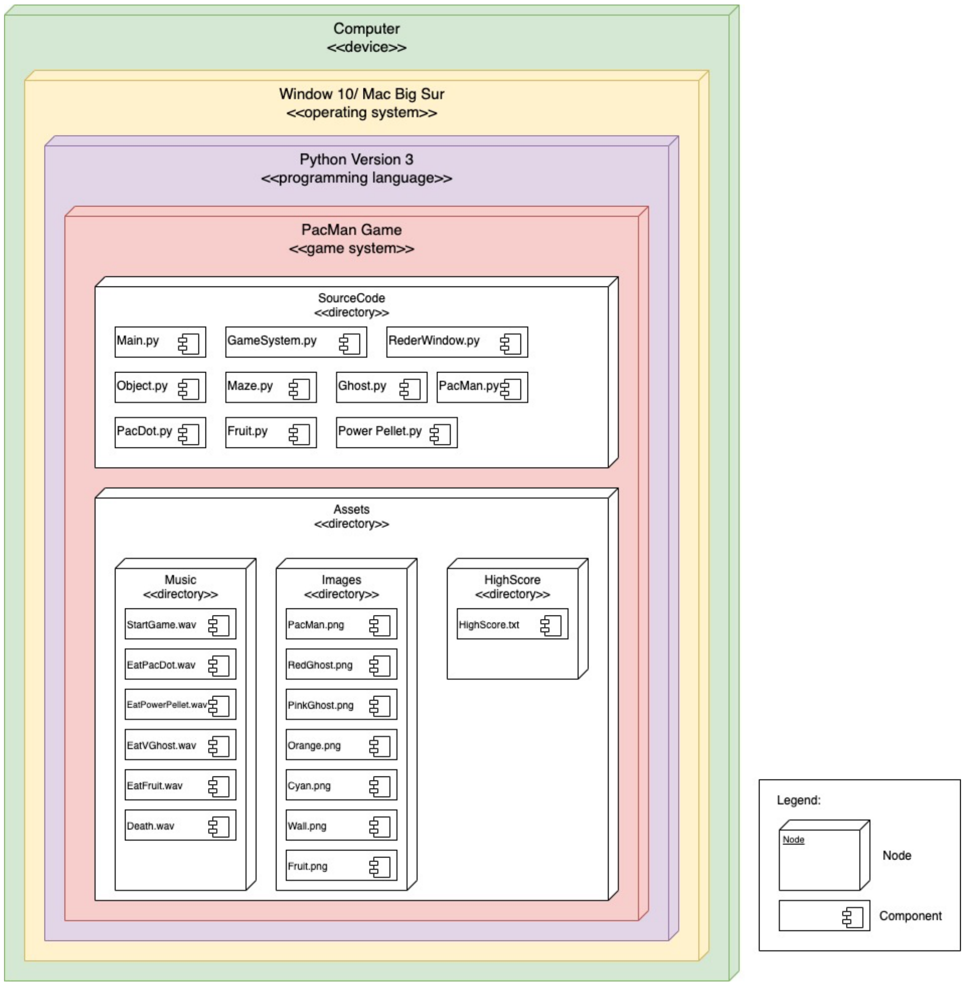


Figure 7: Deployment View for Entire System

4.0 Video link

https://youtu.be/eMp7g86GyKc

Appendix A

Scoring System

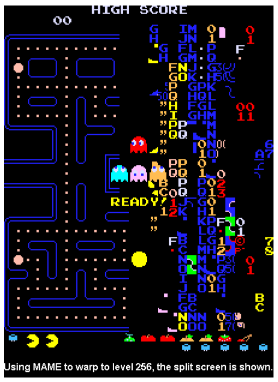
- Pac-Dot - 10 points.
- Power Pellet - 50 points.
- Vulnerable Ghosts:
    - First Succession - 200 points.
    - Second Succession - 400 points.
    - Third Succession - 800 points.
    - Fourth Succession - 1600 points.
- Fruit:
    - 🍒 Cherry: 100 points.
    - 🍓 Strawberry: 300 points
    - 🍊 Orange: 500 points
    - 🍎 Apple: 700 points
    - 🍈 Melon: 1000 points
    - Galaxian Boss: 2000 points
    - 🔔 Bell: 3000 points
    - 🔑 Key: 5000 points

Ghost Mode

Mode	Level 1	Levels 2-4	Levels 5+
Scatter	7	7	5
Chase	20	20	20
Scatter	7	7	5
Chase	20	20	20
Scatter	5	5	5
Chase	20	1033	1037
Scatter	5	1/60	1/60
Chase	indefinite	indefinite	indefinite

Note: Frightening Behavior occurred only when Pac-Man eats the energizer.

Split-Screen Level



Using MAME to warp to level 256, the split screen is shown.

Note: the original Pac-Man system cannot go over level 255.