Assignment
Stage Two 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	05 Oct	30mn	30mn	08 Oct
1.2	Total Working Hours	Heang Sok	15mn	1hr15mn	05 Oct	5mn	35mn	08 Oct
1.3	Effort and Contribution Table	Heang Sok	15mn	1hr30mn	05 Oct	5mn	40mn	08 Oct
1.4	Automatic Documentation	Heang Sok	1hr	2hr30mn	05 Oct	20mn	1hr	08 Oct
2.1	MVC Architectural Design Pattern	Heang Sok	3hr	5hr30mn	1 Sep	3hr	4hr	01 Sep
2.2	Factory Design Pattern	Heang Sok	2hr	7hr30mn	1 Sep	3hr	7hr	01 Sep
2.3	State Design Pattern	Heang Sok	2hr	9hr30mn	1 Sep	3hr	10hr	02 Sep
2.4	Design Tactic	Heang Sok	3hr	12hr30mn	1 Sep	5hr	15hr	10 Sep
2.5	Random Maze Generation	Heang Sok	5hr30mn	18hr	5 Sep	15hr	30hr	20 Sep
2.6	Path Search Algorithms	Heang Sok	5hr	23hr	7 Sep	30hr	60hr	01 Oct
3.1	Software Test Description	Heang Sok	2hr	25hr	10 Sep	3hr	63hr	05 Oct
3.2	Software Test Report	Heang Sok	2hr	27hr	10 Sep	3hr	66hr	05 Oct
4.0	Reflection	Heang Sok	2hr	29hr	11 Sep	2hr	68hr	07 Oct
5.0	Video Link	Heang Sok	30mn	29hr30mn	12 Sep	1hr	69hr	07 Oct
6.0	Finalising Game	Heang Sok	55hr	84hr30mn	19 Sep	70hr	139hr	08 Oct

Table 1: Time Schedule

1.2 Total working hours

Student Name (#ID)	Plan (hours)	Actual (hours)
HeangSOK_s5204340	84hr30mn	139hr
Total working hours	84hr30mn	139hr
Average working hours per person	84hr30mn	139hr

Table 2: Total Working Hours

1.3 Effort and contribution table

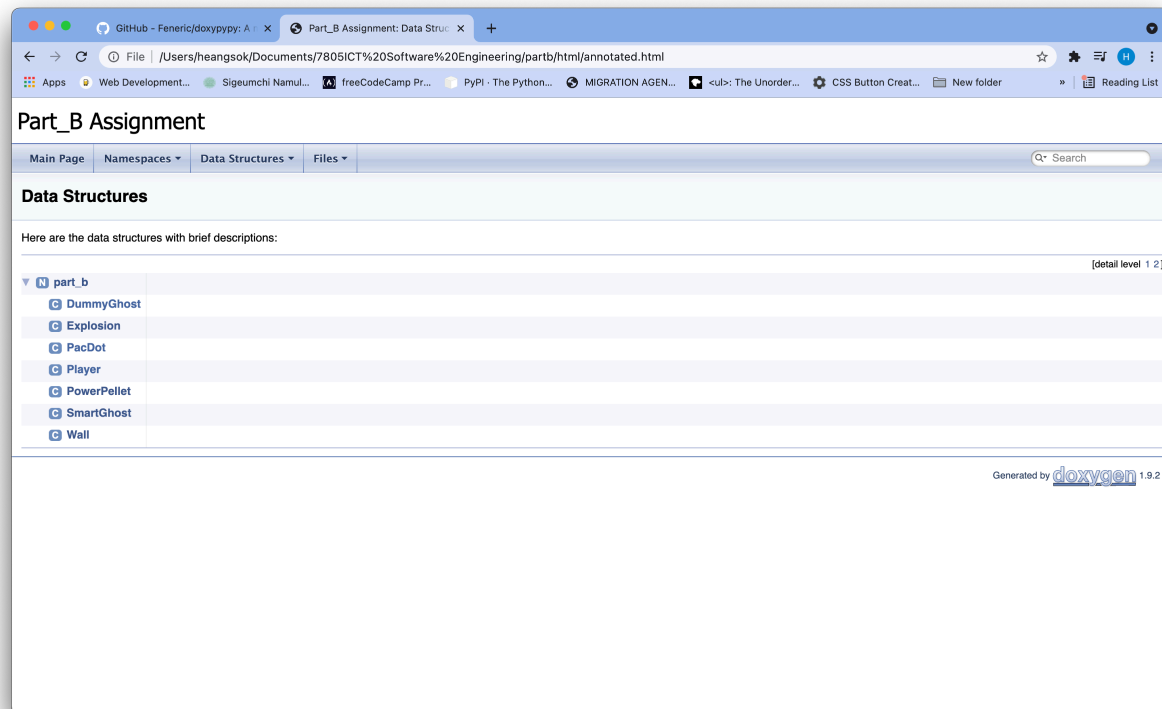
Student	**Effort Level*** (Rating from 0 – 5, the information is filled by the group)	**Contribution Level*** (Rating from 0 – 5, the information is filled by the group)	**Justification** If a student received level rating of 3 or less, your group need to give explanation for the low level rating
Heang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

The Doxygen software is being used to automate document generation as shown in the picture below:



2.0 Advanced Design

2.1 MVC Architectural Design Pattern

2.1.1 Briefly Explain MVC D

According to week 6 lecture slides, the model-view-controller (MVC) pattern separates application (Pacman Game) functionality into three categories:

- **Model**: contains the application's data/state and application logic. One model may have multiple views.

- **View**: is a user interface that produces a representation of the model for the user. Note that sometimes it also has user input function. For example, in some games, the player can create his username account to track the game score or level.

- **Controller**: manages the interaction between the model and the view.

2.1.2 Design Diagrams That Explain How MVC Is Implemented

As shown in figure 1, the player invokes the "Game Controller". Then the "Game Controller" maps the player actions to "Game Model" by changing its state, and it also selects "Game View" for response. Next, the "Game Model" exposes application functionality, responds to state queries, and notifies views of changes. Lastly, the "Game View" renders the models, responds to player gestures, and prints it on the screen for the player to interact.

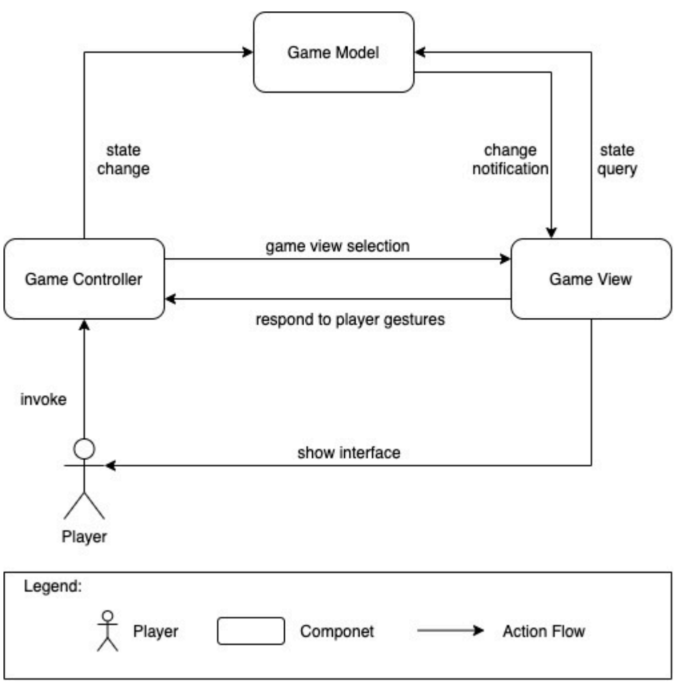


Figure 1: MVC Diagram

The figure 2 below shows the Sequence Diagram which virtualises how the MVC design pattern is being used in the Pacman game. While the game is running, the player controls the Pacman to move forward. Then the GameSystem will calculate all the logics behind the scenes and change it state. The changed state then will be notified and render on the screen.

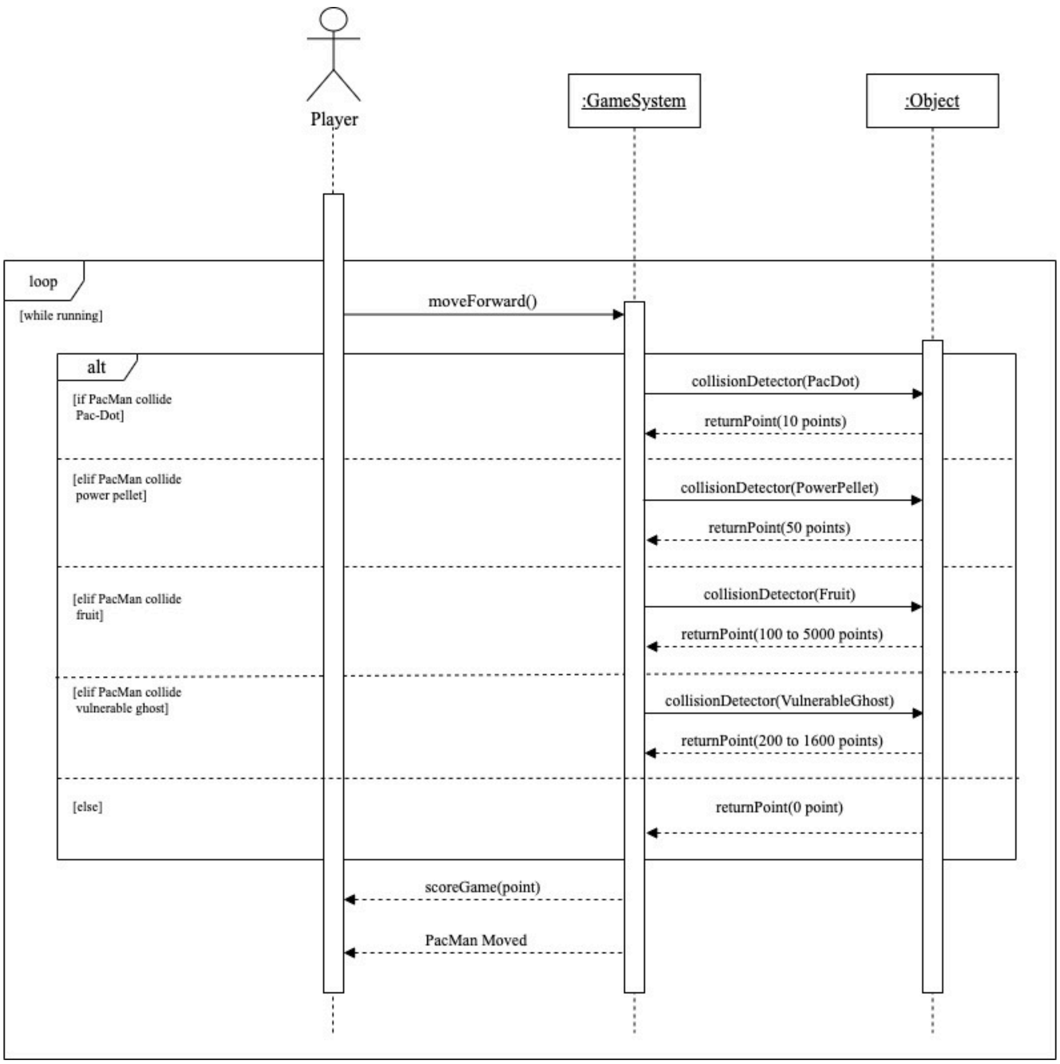


Figure 2: Sequence Diagram

2.1.3 Relevant Source Code That Show How MVC Is Implemented

The source code for Game Controller is illustrated in the below pictures:

```python
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image_origin = pacman_img  # use this technique when rotate many times
        self.image_origin.set_colorkey(BLACK)
        self.image = self.image_origin.copy()
        self.rect = self.image.get_rect()  # second mandatory
        self.rect.x = 14 * SPRITE
        self.rect.y = 23 * SPRITE
        self.speedx = 0  # pygame variable to move the object on x axis
        self.speedy = 0
        self.last_update = pygame.time.get_ticks()
        self.score = 0
        self.lives = 3
        self.pac_dot_sound = pygame.mixer.Sound(os.path.join('asset/music', 'eat.wav'))
        self.power_pellet_sound = pygame.mixer.Sound(os.path.join('asset/music', 'power_pellet.wav'))
        self.hide_timer = pygame.time.get_ticks()
        self.hidden = False

    def mouth_animation(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > 80:
            self.last_update = now
            self.image_origin = pacman_img_close  # use this technique when rotate many times
            self.image_origin.set_colorkey(BLACK)

        else:
            self.image_origin = pacman_img  # use this technique when rotate many times
            self.image_origin.set_colorkey(BLACK)
        ...

    def eat_pac_dots(self):
        hit_pac_dots = pygame.sprite.spritecollide(player, pac_dots, True)
        if hit_pac_dots:
            self.score += 10
            self.pac_dot_sound.play()

    def hide(self):
        self.hidden = True
        self.hide_timer = pygame.time.get_ticks()
        self.rect.center = (WIDTH/2, HEIGHT + 300)

    def update(self):  # when u work with sprite and place the all_sprites.update() in while loop this function will update au
        if self.hidden and pygame.time.get_ticks() - self.hide_timer > 300:
            self.hidden= False
            self.rect.center = player_rect_center
        self.eat_pac_dots()
        self.mouth_animation()
        self.speedx = 0 # turn your speed to 0 here or the player keep running
        self.speedy = 0

        keystate = pygame.key.get_pressed()
        if keystate[pygame.K_a]:
            self.speedx = -5
            new_image = pygame.transform.rotate(self.image_origin, 180)
            self.image = new_image

        if keystate[pygame.K_d]:
            self.speedx = +5
            new_image = pygame.transform.rotate(self.image_origin, 0)
            self.image = new_image
```

```python
if keystate[pygame.K_w]:
    self.speedy = -5
    new_image = pygame.transform.rotate(self.image_origin, 90)
    self.image = new_image

if keystate[pygame.K_s]:
    self.speedy = + 5
    new_image = pygame.transform.rotate(self.image_origin, 270)
    self.image = new_image

self.rect.x += self.speedx
self.rect.y += self.speedy

collide_with_wall = pygame.sprite.groupcollide(player_sprite, walls, False, False)
if collide_with_wall: # this if clause must be used immediately after detect the collision
    self.rect.x += -self.speedx
    self.rect.y += -self.speedy
    self.speedx = 0 # turn your speed to 0 here or the player keep running
    self.speedy = 0

if self.rect.left > WIDTH: # make the player move back to the edge
    self.rect.right = 0 # make the player go to x = 0
if self.rect.right < 0: # make the player move back to the edge
    self.rect.left = WIDTH # make the player go to x = WIDTH
```

The source code for Game Model and Game View are illustrated in the below pictures:

```python
def GameSystem(): # 464
    character()
    vulnerable_state = False
    time_check_point = 0
    # current_time = 0
    # game_over = True
    running = True
    pygame.mixer.music.play()
    while running:

        # keep loop running at the right speed
        clock.tick(FPS)
        # Process input (events)
        for event in pygame.event.get():
            # check for closing window
            if event.type == pygame.QUIT:
                running = False

        current_time = pygame.time.get_ticks()
        collide_with_power_pellet = pygame.sprite.spritecollide(player,power_pellets,True)
        if collide_with_power_pellet:
            time_check_point = pygame.time.get_ticks()
            player.score += 50
            player.power_pellet_sound.play()
            vulnerable_state = True
            red_ghost.image = blue_ghost_img
            pink_ghost.image = blue_ghost_img
            orange_ghost.image = blue_ghost_img
            cyan_ghost.image = blue_ghost_img

        player.eat_pac_dots() #494

        if vulnerable_state and current_time - time_check_point > 2500:
            print(f"mc{current_time}")
            print(f"moT{time_check_point}")
            vulnerable_state = False
            red_ghost.image = red_ghost_img
            pink_ghost.image = pink_ghost_img
            orange_ghost.image = orange_ghost_img
```

```python
        cyan_ghost.image = cyan_ghost_img

    print(f"curr: {current_time},   => checkin time: {time_check_point}" )
    collide_with_dummyghost = pygame.sprite.spritecollide(player, dummy_ghost, False)
    if collide_with_dummyghost and vulnerable_state:
        for hit in collide_with_dummyghost:
            hit.rect.center = orange_rect_center
            player.score += 200
    elif collide_with_dummyghost:
        death_sound_effect.play()
        for hit in collide_with_dummyghost:
            explosion = Explosion(hit.rect.center)
            all_sprites.add(explosion)
            player.hide()
            red_ghost.rect.center = red_rect_center
            pink_ghost.rect.center = pink_rect_center
            orange_ghost.rect.center = orange_rect_center
            cyan_ghost.rect.center = cyan_rect_center
            # player.rect.center = player_rect_center
            player.lives -= 1

    collide_with_smartghost = pygame.sprite.spritecollide(player, smart_ghost, False)
    if collide_with_smartghost and vulnerable_state:
        for hit in collide_with_smartghost:
            hit.rect.center = red_rect_center
            player.score += 200
    elif collide_with_smartghost:
        death_sound_effect.play()
        for hit in collide_with_smartghost:
            explosion = Explosion(hit.rect.center)
            all_sprites.add(explosion)
            player.hide()
            red_ghost.rect.center = red_rect_center
            pink_ghost.rect.center = pink_rect_center
            orange_ghost.rect.center = orange_rect_center
            cyan_ghost.rect.center = cyan_rect_center
            # player.rect.center = player_rect_center
            player.lives -= 1

    if player.lives == 0:
        show_gameover_screen()

    if len(pac_dots) == 0 and len(power_pellets) == 0:
        show_winner_screen()

    # Update
    all_sprites.update()
    # draw/render
    screen.fill(BLACK)
    all_sprites.draw(screen)
    draw_text(screen, f"Level: 01", 18, WIDTH + 25, 5)
    draw_text(screen, f"Lives: ", 18, WIDTH + 25, 30)
    # live_bar(screen, WIDTH + 80, 35, player.lives)
    draw_text(screen, f"Score: {player.score}", 18, WIDTH + 25, 55)
    draw_text(screen, f"High Score: {high_score}", 18, WIDTH + 25, 80)
    draw_lives(player.lives)

    # *after* drawing everything, flip the display
    pygame.display.flip()

pygame.quit()
```

2.2 Factory Design Pattern

2.2.1 Briefly Explain Factory Design Pattern

According to week 6 lecture slides, the purpose of Factory design pattern is to create individual objects in situations where the constructor alone is inadequate.

The reason for using Factory design pattern is because a class can be reused, and the design pattern also lets the class to instantiate numbers of new object as required so that the developers do not need to repeat themselves (write repeating source code).

2.2.2 Design Diagrams That Explain How Factory Design Pattern Is Implemented

The class diagram below shows how the Factory Design Pattern is being used in the Pacman Game. In this game, we will have 2 dummy ghosts which are Orange and Cyan Ghost. These dummy ghosts will move randomly through the maze. Note that all the logics to make these ghosts to move randomly is placed in a function called update because in python when working with the pygame.sprite library, it is mandatory to place all the main logics in the update function (we cannot name the function to another name) or the code will not run as it is supposed to be.

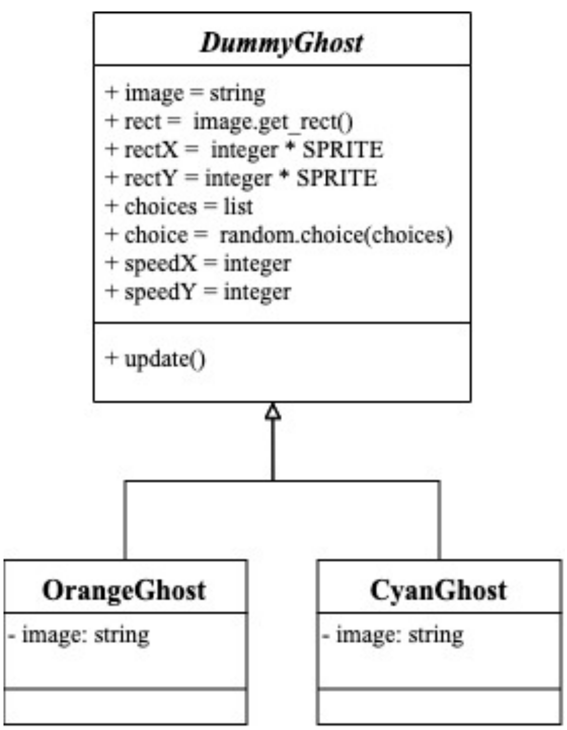


Figure 3: Class Diagram that show the implementation of the Factory Design Pattern

2.2.3 Relevant Source Code That Show How Factory Design Pattern Is Implemented

To implement the Factory Design Pattern, we first create a reusable class for the dummy ghosts.

```python
class DummyGhost(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = orange_ghost_img  # first mandatory
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()  # second mandatory
        self.rect.x = random.randrange(11, 15) * SPRITE
        self.rect.y = random.randrange(13, 15) * SPRITE
        self.choices = [[-4,4], [-3,3]]
        self.choice = random.choice(self.choices)
        self.speedx = random.choice(self.choice) # pygame variable to move the object on x axis
        self.speedy = random.choice(self.choice)

    def update(self):
        self.rect.y += self.speedy
        collide_with_wall = pygame.sprite.groupcollide(dummy_ghost,walls,False,False)
        if collide_with_wall:
            self.rect.y += -self.speedy
            self.rect.x += self.speedx
            # self.speedy = random.choice(self.choice) # note: if you put it here, it wont work
            # self.speedx = random.choice(self.choice) # note: if you put it here, it wont work
            collide_with_wall = pygame.sprite.groupcollide(dummy_ghost,walls,False,False)
            if collide_with_wall:
                self.rect.x += -self.speedx
                self.speedy = random.choice(self.choice)
                self.speedx = random.choice(self.choice)
```

Then we instantiate the orange and cyan ghost objects.

```python
    # --------------------------------dummy_ghost
    dummy_ghost = pygame.sprite.Group()
    orange_ghost = DummyGhost()
    orange_ghost.image = orange_ghost_img
    all_sprites.add(orange_ghost)
    dummy_ghost.add(orange_ghost)
    orange_rect_center = orange_ghost.rect.center

    cyan_ghost = DummyGhost()
    cyan_ghost.image = cyan_ghost_img
    all_sprites.add(cyan_ghost)
    dummy_ghost.add(cyan_ghost)
    cyan_rect_center = cyan_ghost.rect.center
```

2.3 State Design Pattern

2.3.1 Briefly Explain State Design Pattern

According to week 7 lecture slides, the purpose of the State Design Pattern is to cause an object to behave in a manner determined by its state.
The reason for using this pattern is because when the Pacman eats the Power-Pellet, it causes all the ghosts to go into the vulnerable state. When the ghosts are in vulnerable state, the Pacman can eat them to get more scores.

2.3.2 Design Diagrams That Explain How State Design Pattern Is Implemented

The figure 4 shows that when the Pacman eat a power pellet, it will cause all the ghosts to go into the vulnerable state, and the ghosts will become normal again after 2.5 seconds.

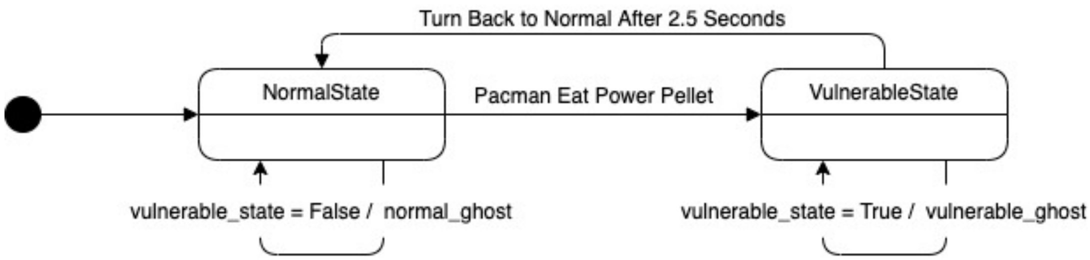


Figure 4: State Diagram that show the implementation of the State Design Pattern

2.3.3 Relevant Source Code That Show How The State Design Pattern Is Implemented

The source code below shows how the ghosts change from normal to vulnerable state and turn back to normal state again.

```python
def GameSystem():
    character()
    vulnerable_state = False
    time_check_point = 0
    running = True
    pygame.mixer.music.play()
    while running:

        # keep loop running at the right speed
        clock.tick(FPS)
        # Process input (events)
        for event in pygame.event.get():
            # check for closing window
            if event.type == pygame.QUIT:
                running = False

        current_time = pygame.time.get_ticks()
        collide_with_power_pellet = pygame.sprite.spritecollide(player,power_pellets,True)
        if collide_with_power_pellet:   # Ghost become vulnerable and turn blue
            time_check_point = pygame.time.get_ticks()
            player.score += 50
            player.power_pellet_sound.play()
            vulnerable_state = True
            red_ghost.image = blue_ghost_img
            pink_ghost.image = blue_ghost_img
            orange_ghost.image = blue_ghost_img
            cyan_ghost.image = blue_ghost_img

        if vulnerable_state and current_time - time_check_point > 2500:  # 2500 = 2.5s
            vulnerable_state = False  # after 2.5 second, the ghost will become normal and turn back to its original color
            red_ghost.image = red_ghost_img
            pink_ghost.image = pink_ghost_img
            orange_ghost.image = orange_ghost_img
            cyan_ghost.image = cyan_ghost_img
```

2.4 Design tactic

2.4.1 Explain Design Tactics and Quality Attributes

According to the week 8 lecture slides, a tactic is a design decision that influences the achievement of a quality attributes, and the quality attribute (QA) is a testable property of a system that is used to check whether the system works well or not. The quality attribute includes availability, modifiability, performance, security, testability, and usability.

The usability tactic has been applied in this project. According to the week 8 lecture slides, the usability tactic focuses on how easy it is for the player to understand how to play the game. This tactic improves the game usability by:

- **Learning the system features**: The game's labels and buttons are written in English to make it easy for the player to learn how to play.
- **Using the system efficiently**: The system will run on the local machine; thus, there is no downtime due to the internet. The system can also run across platforms (Mac or Window).
- **Increasing satisfaction**: The game comes with high resolution graphical interface and sound. The game also gives the player a challenging environment by running from the ghost and scoring from Pac-Dots

2.4.2 Usability Tactic General Scenario and Diagram

Portion of Scenario	Possible Values
Source:	Player
Stimulus:	Player wants to learn how to play the game
Artifact:	System
Environment:	At runtime
Response:	**Learning system feature**: The game's labels and buttons are written in English to make it easy for the player to learn how to play (The game start-up page's source code will be provided in the section 2.4.3).
Response Measure:	Player knows how to play

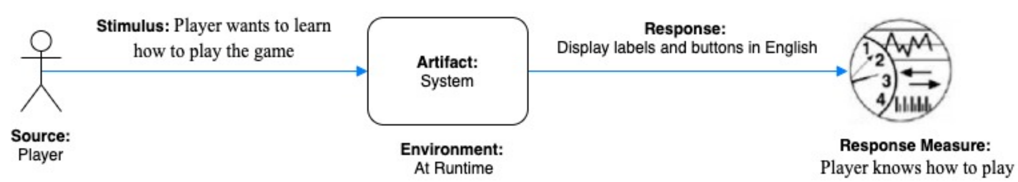


Figure 5: Usability Tactic Scenario

2.4.3 Relevant Source Code That Show Usability Tactic Is Implemented

The following source code makes the game system to display labels and buttons of the start-up page in English so that the player can learn to play easily.

```python
def start_up():
    running = True
    click = False
    while running:
        # keep loop running at the right speed
        clock.tick(FPS)
        screen.fill(BLACK)
        pos = pygame.mouse.get_pos()
        # Process input (events)
        for event in pygame.event.get():
            # check for closing window
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                click = True

        pb = play_button.get_rect()
        pb.x, pb.y = (WIDTH + 100) // 2, 430+25+35
        ob = option_button.get_rect()
        ob.x, ob.y = (WIDTH + 100) // 2, 520+25+35
        qb = quit_button.get_rect()
        qb.x, qb.y = (WIDTH + 100) // 2, 610+25+35
        if pb.collidepoint(pos): #play_button
            if click:
                game()
                click = False
        elif ob.collidepoint(pos): #option_button
            if click:
                maze_configuration()
                click = False
        elif qb.collidepoint(pos): #quit_button
            if click:
                quit()
                click = False
        # render/draw
        screen.blit(logo1_img, ((WIDTH + 80) // 2, 5+25))
        screen.blit(logo_img,((WIDTH + 80)//2, 75+25))
        screen.blit(play_button, (pb.x, pb.y))
        screen.blit(option_button, (ob.x, ob.y))
        screen.blit(quit_button, (qb.x, qb.y))
        draw_text(screen, f"Title: ", 20, (WIDTH + 80)//2, 285+25)
        draw_text(screen, f"YEAR:", 20, (WIDTH + 80)//2, 285+25+35)
        draw_text(screen, f"COURSE:", 20, (WIDTH + 80) // 2, 320+25+35)
        draw_text(screen, f"NAME:", 20, (WIDTH + 80) // 2, 355+25+35)
        draw_text(screen, f"SNUMBER:", 20, (WIDTH + 80) // 2, 390+25+35)
        draw_text(screen, f"Assignment 1", 20, (WIDTH + 330) // 2, 285 + 25)
        draw_text(screen, f"2021", 20, (WIDTH + 340)//2, 285+25+35)
        draw_text(screen, f"7805ICT", 20, (WIDTH + 340) // 2, 320+25+35)
        draw_text(screen, f"Heang_Sok", 20, (WIDTH + 340) // 2, 355+25+35)
        draw_text(screen, f"s5204340", 20, (WIDTH + 340) // 2, 390+25+35)

        pygame.display.flip()

    pygame.quit()

start_up()
```

2.5 Random maze generation

To randomly generate a maze, the **Recursive Backtracker Algorithm** is being used in this project. According to the Wikipedia, it is necessary to follow all the steps below:

- Step 1: Pick a starting cell and mark it as visited
- Step 2: If any neighboring cell has not been visited:
    - Pick a random neighboring cell that hasn't been visited
    - Remove the wall between the two cells
    - Add the current cell to the stack
    - Make the chosen cell the current cell and mark it as visited
- Step 3: If the current cell has no unvisited neighbors, take the top cell from the stack, and make it current
- Step 4: Repeat from #2 until there are no more unvisited cells

The figure 6 shows the algorithm of how the system generates the random maze.

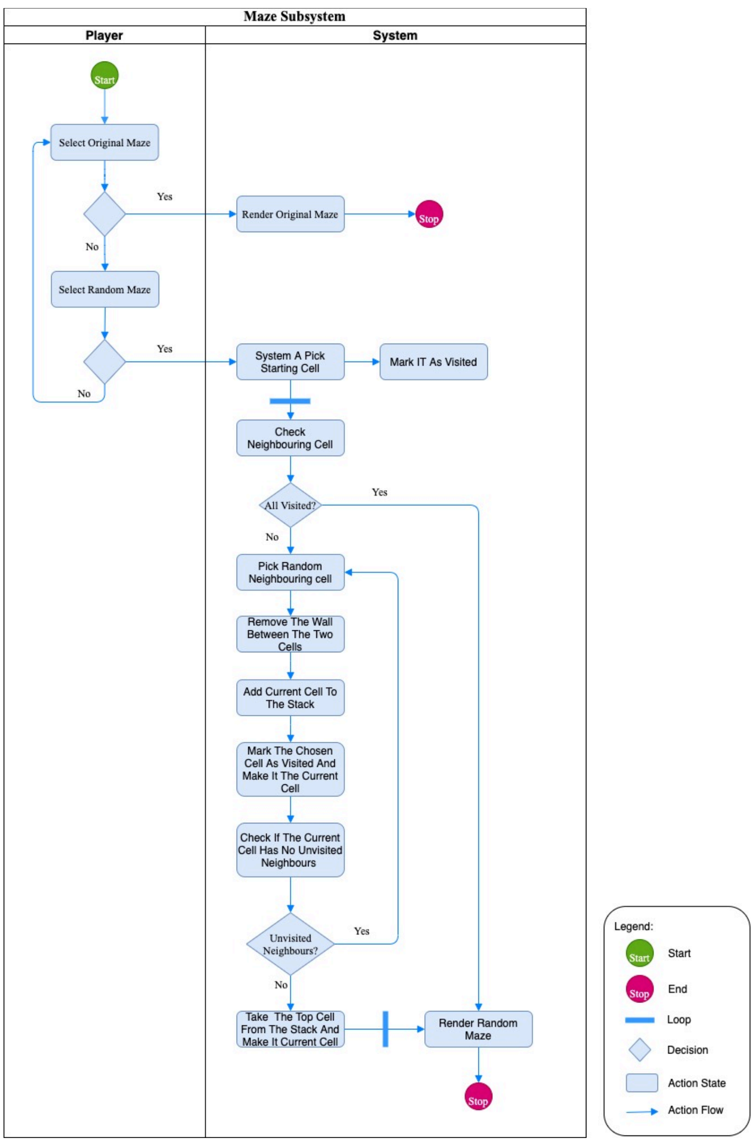


Figure 6: Activity Diagram for Maze Subsystem

The relevant source code for random maze generator is shown in the pictures below:

```python
board = [
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 6, 2, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 6, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 2, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 2, 2, 2, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 4, 4, 4, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 3],  # Middle Lane
    [3, 3, 3, 3, 3, 3, 2, 3, 3, 1, 3, 1, 1, 4, 1, 1, 1, 3, 1, 3, 3, 2, 3, 3, 3, 3, 3, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 6, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 1, 1, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 6, 3],
    [3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3],
    [3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
]
```

```python
# predefine the location that we want to ignore
visited = [[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [0, 10], [0, 11], [0, 12],
           [0, 13], [0, 14], [0, 15], [0, 16], [0, 17], [0, 18], [0, 19], [0, 20], [0, 21], [0, 22], [0, 23], [0, 24],
           [0, 25], [0, 26], [0, 27], [30, 0], [30, 1], [30, 2], [30, 3], [30, 4], [30, 5], [30, 6], [30, 7], [30, 8],
           [30, 9], [30, 10], [30, 11], [30, 12], [30, 13], [30, 14], [30, 15], [30, 16], [30, 17], [30, 18], [30, 19],
           [30, 20], [30, 21], [30, 22], [30, 23], [30, 24], [30, 25], [30, 26], [30, 27], [1, 0], [1, 27], [2, 0],
           [2, 27], [3, 0], [3, 27], [4, 0], [4, 27], [5, 0], [5, 27], [6, 0], [6, 27], [7, 0], [7, 27], [8, 0], [8, 27],
           [9, 0], [9, 27], [10, 0], [10, 27], [11, 0], [11, 27], [12, 0], [12, 27], [13, 0], [13, 27], [14, 0],
           [14, 27], [15, 0], [15, 27], [16, 0], [16, 27], [17, 0], [17, 27], [18, 0], [18, 27], [19, 0], [19, 27],
           [20, 0], [20, 27], [21, 0], [21, 27], [22, 0], [22, 27], [23, 0], [23, 27], [24, 0], [24, 27], [25, 0],
           [25, 27], [26, 0], [26, 27], [27, 0], [27, 27], [28, 0], [28, 27], [29, 0], [29, 27],[23, 13], [23, 14],
           [3, 1], [3, 26], [23, 1], [23, 26],[14, 12], [14, 13], [14, 14], [15, 13],
           ]
```

```python
x = 1
y = 1
start_position = [x, y]
stack = []
grid = []
stack.append([x, y])
visited.append(start_position)

# define the grid
for i, row in enumerate(board):
    for j, block in enumerate(row):
        if [i, j] not in visited:
            grid.append([i, j])
```

```python
def randomMaze():
    while len(stack) > 0:
        move = []
        if [x + 2, y] not in visited and [x + 2, y] in grid:  # prevent the calculation go off the screen
            move.append("right")
        if [x - 2, y] not in visited and [x - 2, y] in grid:
            move.append("left")
        if [x, y + 2] not in visited and [x, y + 2] in grid:
            move.append("down")
        if [x, y - 2] not in visited and [x, y - 2] in grid:
            move.append("up")

        if len(move) > 0:
            random_move = random.choice(move)  # pick a random neighboring cell that hasn't been visited
            if random_move == "right":
                x += 2
                board[x-1][y] = 2  # Remove the wall
                stack.append([x, y])  # add the current cell to the stack
                visited.append([x, y])  # make the chosedn cell the current cell and mark it visited

            elif random_move == "left":
                x -= 2
                board[x+1][y] = 2  # Remove the wall
                stack.append([x, y])  # add the current cell to the stack
                visited.append([x, y])  # make the chosedn cell the current cell and mark it visited

            elif random_move == "down":
                y += 2
                board[x][y-1] = 2  # Remove the wall
                stack.append([x, y])  # add the current cell to the stack
                visited.append([x, y])  # make the chosedn cell the current cell and mark it visited

            elif random_move == "up":
                y -= 2
                board[x][y+1] = 2  # Remove the wall
                stack.append([x, y])  # add the current cell to the stack
                visited.append([x, y])  # make the chosedn cell the current cell and mark it visited
        else:
            x, y = stack.pop()  # if the current cell has no unvisited neighbors, take the top cell from the stack and
            # make it visited

random = randomMaze()
```

2.6 Path search algorithms

2.6.1 Dummy Ghost AI

In this project, there are two dummy ghosts moving randomly through the maze. To achieve this behaviour the following algorithms is used:

- Step 1: Select a random movement on x axis (right or left)
- Step 2: Select a random movement on y axis (up or down)
- Step 3: Move according to the random movement
- Step 4: If the dummy ghost collides with the wall, it will stop and move in a new random direction. If the dummy ghost does not collide with the wall, it continues to move forward based on its current direction.

Figure 7 shows the algorithm of how the dummy ghosts move.

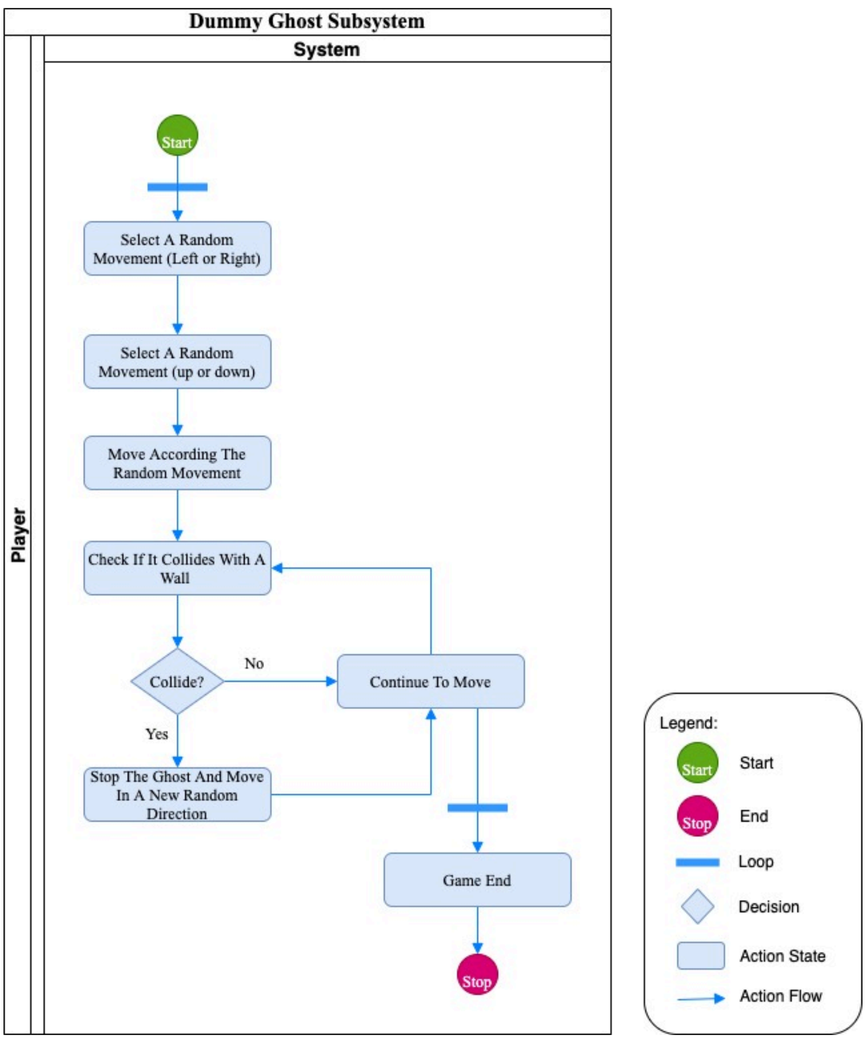


Figure 7: Activity Diagram for Dummy Ghost Subsystem

The relevant source code for dummy ghosts' movement:

```python
class DummyGhost(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = orange_ghost_img  # first mandatory
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()  # second mandatory
        self.rect.x = random.randrange(11, 15) * SPRITE
        self.rect.y = random.randrange(13, 15) * SPRITE
        self.choices = [[-4,4], [-3,3]]
        self.choice = random.choice(self.choices)
        self.speedx = random.choice(self.choice) # pygame variable to move the dummy ghost on x axis
        self.speedy = random.choice(self.choice) # pygame variable to move the dummy ghost on y axis

    def update(self):
        self.rect.y += self.speedy  # move the ghost
        collide_with_wall = pygame.sprite.groupcollide(dummy_ghost,walls,False,False)
        if collide_with_wall:
            self.rect.y += -self.speedy  # stop the ghost
            self.rect.x += self.speedx  # move in new direction
            # self.speedy = random.choice(self.choice) # note: if you put it here, it wont work
            # self.speedx = random.choice(self.choice) # note: if you put it here, it wont work
            collide_with_wall = pygame.sprite.groupcollide(dummy_ghost,walls,False,False)
            if collide_with_wall:
                self.rect.x += -self.speedx
                self.speedy = random.choice(self.choice)
                self.speedx = random.choice(self.choice)
```

2.6.1 Smart Ghost AI

In this project, there are two smart ghosts chase after the Pacman. To achieve this behaviour the Breadth First Search (BFS) algorithm is used:

- Step 1: Define a function take the Pacman's x coordinate and y coordinate as the goal
- Step 2: Define a ghost starting position
- Step 3: Mark the starting position as visited
- Step 4: Make a queue and enqueue the starting position
- Step 5: Define a while loop to check all the possible path (loop will stop when the queue is empty):
    + Start to dequeue using "First In First Out" mechanism
    + If the dequeue element is the goal, then return the dequeue element and start the next iteration
    + Loop all the adjacent nodes; if the next node is not the wall and visited, enqueue the node position, mark it as visited, and go to the next iteration

Figure 8 shows the algorithm of how the smart ghosts move.

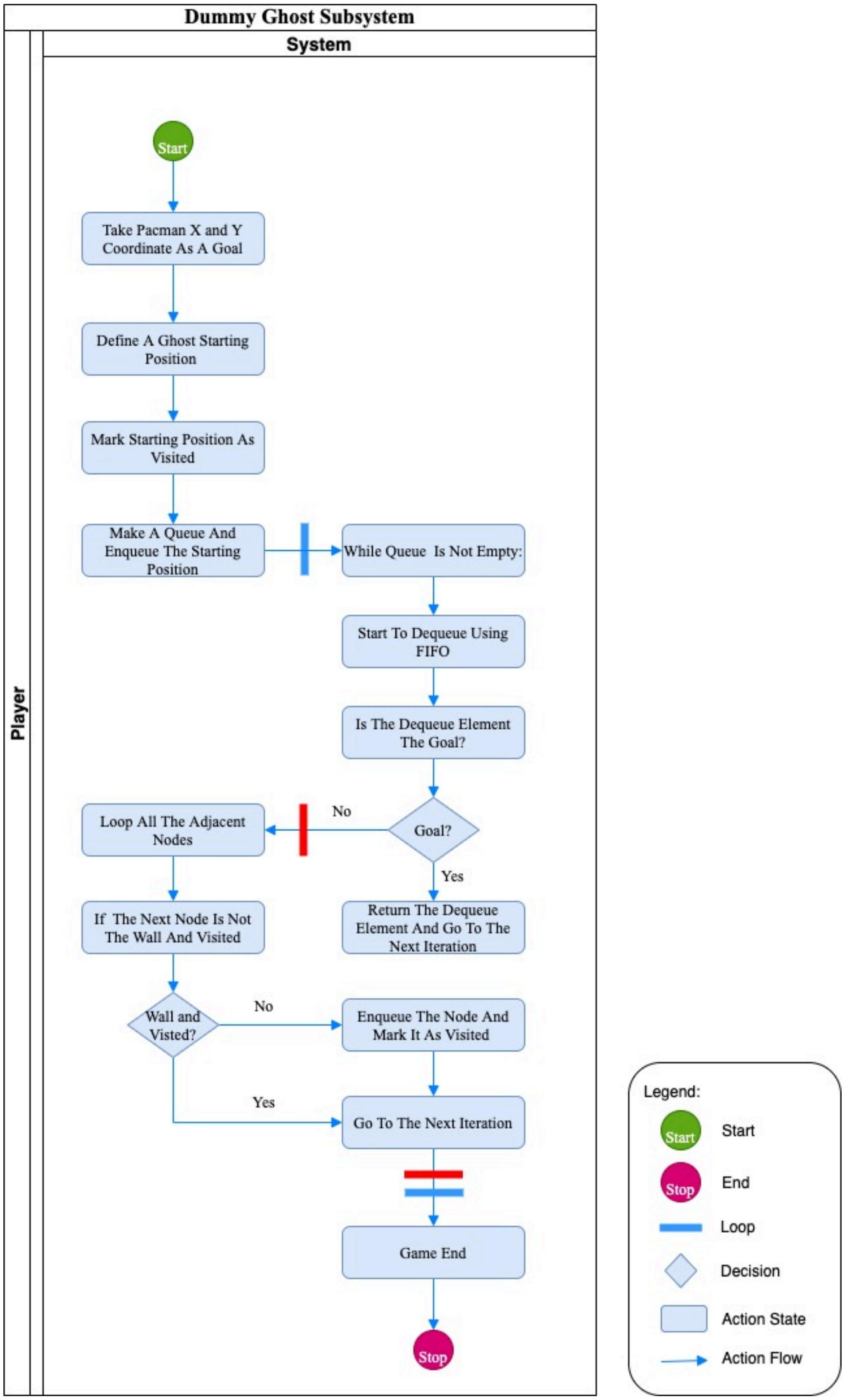


Figure 8: Activity Diagram for Smart Ghost Subsystem

The relevant source code for Smart ghosts' movement:

```python
class SmartGhost(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = red_ghost_img  # first mandatory
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()  # second mandatory
        self.x = random.randrange(11, 15)
        self.y = random.randrange(13, 15)
        self.rect.x = self.x * SPRITE
        self.rect.y = self.y * SPRITE
        self.available_path = []  # available_path prevents the smart ghost not to go through the wall
        # define available_path
        for i, row in enumerate(GAME_BOARD):
            for j, block in enumerate(row):
                if block != 3:
                    self.available_path.append([i,j])

    def breath_first_search(self,goal):
        """Make all the smart ghosts chase after Pacman"""
        # element for breath first search algorithm starts here
        starting_pos = [self.rect.x, self.rect.y]
        visited_node = [starting_pos] # make the starting point as visited
        ghosts_queue = queue.Queue()  # make a blank queue to work with breath first search
        ghosts_queue.put(starting_pos)  # enqueue the starting position

        while ghosts_queue.qsize() != 0:  # stop the loop when the queue is empty
            chasing_path = ghosts_queue.get()  # dequeue
            if chasing_path == goal:
                return chasing_path
            for i in self.available_path:
                # use pygame to check the adjacent node
                collide_with_wall = pygame.sprite.groupcollide(smart_ghost, walls, False, False)
                if not collide_with_wall and chasing_path not in visited_node:
                    visited_node.append(chasing_path)
                    ghosts_queue.put(i)

    ...
```

3.0 Testing

3.1 Software test description

3.1.1 Test Environment

Since the Pacman system is run on the local machine, it is important to have an appropriate Hardware and Software in order to do the software testing. Table 4 provides a full descriptions of test environment that is required for this project.

Test Environment	Requirements
Hardware	• Standard computer machine requirements for this project: ○ Processor: 1 gigahertz (GHz) or faster processor ○ RAM: 4GB ○ Storage: 32GB ○ I/O devices: keyboard, mouse, monitor, and speaker
Operating System	• Window machine: Window8 or Window10 • Mac machine: Mac OS X or Mac OS Big Sur
Programming Language	Python Version 3+
Integrated Development Environment (IDE)	PyCharm version 2021.2
Python Package Library	pygame, random, queue, math

Table 4: Detailed Description of the test environment

3.1.2 Build Test Cases

Test Case 01: The Pacman can move in the maze based on key control.

R1: if the player presses key "a", the Pacman moves left
R2: if the player presses key "d", the Pacman moves right
R3: if the player presses key "w", the Pacman moves up
R4: if the player presses key "s", the Pacman moves down

PROJECT:		Pacman System		
MODULE:		PacMan.py		
REQUIREMENT:		R1, R2, R3, R4		
TEST CASE ID:		TC_01		
TEST OBJECTIVE:		To test whether the Pacman moves based on key control or not		
TEST DATE AND TIME		05/10/2021		
<u>Step No</u>	**<u>Remarks</u>**	**<u>Key input</u>**	**<u>Expected Results</u>**	**<u>Actual Results</u>**
1	if the player presses key "a"	key = a	Pacman moves left	Pacman moves left
2	if the player presses key "d"	key = d	Pacman moves right	Pacman moves right
3	if the player presses key "w"	key = w	Pacman moves up	Pacman moves up
4	if the player presses key "s"	key = s	Pacman moves down	Pacman moves down

Table 5: Test case 01

Test Case 02: The Pacman is killed by a ghost function is successfully implemented.
R1: Pacman has 3 lives, when it captured by a ghost; lives minus1
R2: Another Pacman will appear, and all ghosts will go back to their base.
R3: When there are no more lives, the game will be over and display "Game - Over"

PROJECT:	Pacman System			
MODULE:	PacMan.py			
REQUIREMENT:	R1, R2, R3			
TEST CASE ID:	TC_02			
TEST OBJECTIVE:	To test the Pacman is killed by a ghost function is successfully implemented or not			
TEST DATE AND TIME	05/10/2021			
Step No	**Steps**	**Lives**	**Expected Results**	**Actual Results**
1	Pacman collides with a ghost for the first time	lives = 3 -1	-The lives minus one -Another Pacman appears -Ghosts go back to their base	-The lives minus one -Another Pacman appears -Ghosts go back to their base
2	Pacman collides with a ghost for the second time	lives = 2 -1	-The lives minus one -Another Pacman appears -Ghosts go back to their base	-The lives minus one -Another Pacman appears -Ghosts go back to their base
3	Pacman collides with a ghost for the third time	lives = 1 -1	-Game Over -Display "Game Over" on the screen	-Game Over -Display "Game Over" on the screen

Table 6: Test case 02

Test Case 03: In the start-up page, the configure button works. The player can select two game - modes: original maze and random maze.
R1: Render original maze on the screen when the player selects original maze.
R2: Render random maze on the screen when the player selects random maze.

PROJECT:		Pacman System		
MODULE:		PacMan.py		
REQUIREMENT:		R1, R2		
TEST CASE ID:		TC_03		
TEST OBJECTIVE:		To test if the maze system renders the selected maze or not		
TEST DATE AND TIME		05/10/2021		
Step No	**Remarks**	**Boolean**	**Expected Results**	**Actual Results**
1	If the play selects the original mode	originalMaze = True	-render original maze	-render original maze
2	If the player selects the random mode	randomMaze = True	-render random maze	-render random maze

Table 7: Test case 03

3.2 Software test report

3.2.1 Test Results

3.2.1.1 Tests Log

This Pacman Game was tested on the Mac machine with the operation system called Big Sur from 1st October 2021 to 5th October 2021.
Tester name: Heang Sok.

3.2.1.2 Rationale for Decision

After executing a test, the decision is defined according to the following rules:
- **Ok**: The test sheet is set to "OK" state when all steps are in "OK" state. The real result is compliant to the expected result.
- **NOK**: The test sheet is set to "NOK" state when all steps of the test are set to "NOK" state or when the result of a step differs from the expected result.
- **Partial OK**: The test sheet is set to "Partial OK" state when at least one step of the test is set to "NOK" state or when the result of a step is partially compliant to the expected result.
- **Not Run**: Default state of a test sheet not yet executed.
- **Not Completed**: The test sheet is set to "Not Completed" state when at least one step of the test is set "Not Run" state.

3.2.1.3 Overall Assessment of Tests

Test Case 01: -All tests with the Pacman movement passed; the players can control Pacman from their keyboard.

Test Case 02: -The Pacman is killed by a ghost function is successfully implemented and passed.
-Live deduction function passed. When the Pacman collides with the ghost, it dies and one live will be deducted.
-Game Over function passed. When there are no more lives left, the game is over.
-Display “Game Over” function passed. When the game is over, the system render “Game Over” text on the screen.

Test Case 03: -Maze configuration function passed. Players can configure their maze by clicking the configure button.
-Render original maze function passed. When a player selects the original maze, the system renders the original maze
- Render random maze function passed. When a player selects the random maze, the system renders the random maze

3.2.1.4 Impact of Test Environment

If the player run this game on a Mac Machine, it will take around 30 seconds to start the game system. However, if the player run this game on a Window Machine, it will take around 5 seconds to start the game.

3.2.2 Summary Table

Test Case 01

Test Case ID: TC_01		**Comment**	**Decision**
Test Description	To test whether the Pacman moves based on key control or not	N/A	OK
Requirement	R1, R2, R3, R4	N/A	N/A
Initial Conditions	The Pacman is waiting for the player's command	N/A	N/A
Tests Inputs	The player press "a"	N/A	N/A
Tests Outputs	The Pacman moves left	N/A	N/A
Expected Result	The Pacman moves left	N/A	N/A
Test Procedure			
Step Number	**Operator Actions**	**Expected Result**	**Result**
1	if the player presses key "a"	the Pacman moves left	OK
2	if the player presses key "d"	the Pacman moves right	OK
3	if the player presses key "w"	the Pacman moves up	OK
4	if the player presses key "s"	the Pacman moves down	OK

Table 8: Summary Table 1

Test Case 02

Test Case ID: TC_02		**Comment**	**Decision**
Test Description	To test the Pacman is killed by a ghost function is successfully implemented or not	N/A	OK
Requirement	R1, R2, R3	N/A	N/A
Initial Conditions	The Pacman has 3 lives	N/A	N/A
Tests Inputs	The Pacman collides with a ghost	N/A	N/A
Tests Outputs	The Pacman's lives minus 1	N/A	N/A
Expected Result	The Pacman's lives minus 1	N/A	N/A
Test Procedure			
Step Number	**Operator Actions**	**Expected Result**	**Result**
1	Pacman collides with a ghost for the first time	-The lives minus one -Another Pacman appears -Ghosts go back to their base	OK
2	Pacman collides with a ghost for the second time	-The lives minus one -Another Pacman appears	OK

		-Ghosts go back to their base	
3	Pacman collides with a ghost for the third time	-Game Over -Display "Game Over" on the screen	OK

Table 9: Summary Table 2

Test Case 03

Test Case ID: TC_03		**Comment**	**Decision**
Test Description	To test if the maze system renders the selected maze or not	N/A	OK
Requirement	R1, R2	N/A	N/A
Initial Conditions	Start-up page is shown	N/A	N/A
Tests Inputs	The player selects original maze	N/A	N/A
Tests Outputs	Render original maze on the screen	N/A	N/A
Expected Result	Render original maze on the screen	N/A	N/A
Test Procedure			
Step Number	**Operator Actions**	**Expected Result**	**Result**
1	If the play selects the original mode	-render original maze	OK
2	If the player selects the random mode	-render random maze	OK

Table 10: Summary Table 3

4.0 Reflection

Coding is a challenging skill, especially for me who was an accountant for 5 years, because we need put a lot of times and efforts into it in order to become a good programmer. Up until now I have learned how to code for more than six months, and Python is the only programming language that I know. This is the reason why I used Python to build the Pacman Game from scratch.

Since I have just learned Python, making the Pacman Game is a very tough and challenging assignment for me, especially when I do this project alone. I decided to do this project alone because I wanted to put pressures on myself so that I would improve my coding skills faster. I also do the same for other group projects in other courses. So, in this trimester, I have to build the Pacman game, one Data Analytical software, and one website alone. However, I was wrong and underestimate those assignments. Even I put more than 10 hours every day on coding, I hardly finished them as some algorithms are too difficult to understand. For example, the **Recursive Backtracker Algorithm** that is used for random maze generation. At some points, I could not bear with the study loads and became so stress.

Reflect upon the experience that I gained from these projects, I learnt how to overcome my stress later on and finished all the projects one by one. I can tell that my coding skills have improved in many ways because my code is looked cleaner and easier to understand. All the hours that I put into building the Pacman project as well as other projects in this trimester make me become skilful in Python Programming Language. Another soft skill that I gained from doing these projects is to go into "Deep Work" mode. According to Dr. Cal Newport, when we are in the "Deep Work" mode, we can remove all the distraction from our mind and focus on our work 100 percent to boost the productivities. This means that whenever I code, I forget about time and my surrounding; all I care is to make the code runs as it is required. This is the reason why I can sit for 10 hours straight just to code. On the other hand, I personally think that I have built more confidence in myself after completing all the projects. Before, I usually doubted about my coding skills and thought that I might not be able to make any program or software at all, but now I made two software and one website from scratch. I believe that I can become a better programmer and make good money from my skill after I graduate.

The last thing that I want to mention here is I would like to express my gratitude for the contribution that the teaching team has made. During the past 11 weeks, the teaching team is always here for every student when they need supports on the assignment as well as the course lessons. I could reflect that having a good teaching team like this course is one of my exciting experiences here as an international student, and I am so glad that I enrolled in this course because the course materials are rich and interesting. More importantly, I could apply the knowledge that I learnt from this course in future work.

5.0 Video link

https://youtu.be/x4n_LWdwU-c