Deliverable 3 – Kayla Cheung

GitHub Link: <https://github.com/kaylacheung/Space-Shooter-Game.git>

1. Core Functions for Prototype

* **Game Loop** (Handles updates, rendering, input)
* **Player Movement** **(SpaceShip Movement)** (allows player to move left right)
* **Player Shooting** (lets the player shoot projectiles)
* **Enemy Spawning** (spawn basic enemies e.g., aliens/ asteroids)
* **Collision Detection** (detects collisions between player projectiles and enemies
* **Basic Rendering** (displays the spaceship, enemies, and projectiles on screen)

1. Prototype Plan

**Classes:**

* **SpaceShip**: Moves left and right using arrow keys/ shoots projectiles using spacebar
* **Enemies**: aliens that move horizontally and wrap around the screen/ spawn in waves at regular intervals
* **Projectile**: represents spaceship’s projectile
* **Game Loop**: Continuously updates and renders the game/ H=handles user input (moving and shooting)
* **Collision Detection**: projectiles destroy enemies when make contact/ enemies damage the player health when make contact
* **Rendering:** use Pygame to draw objects (spaceship, enemies, and projectiles on the screen) and display the game state (score/ lives counter)

1. Prototype Codes

**Spaceship.py:**

1. import pygame

2. from prototype\_projectiles import Projectile

3.

4. class Spaceship(pygame.sprite.Sprite):

5.     def \_\_init\_\_(self):

6.         super().\_\_init\_\_()

7.         self.image = pygame.Surface((50, 50))

8.         self.image.fill((0, 255, 0))  # Green square for player

9.         self.rect = self.image.get\_rect()

10.         self.rect.center = (400, 550)  # Start at the bottom center

11.         self.speed = 5

12.

13.     def update(self, keys):

14.         """Update player position based on key presses."""

15.         if keys[pygame.K\_LEFT] and self.rect.left > 0:

16.             self.rect.x -= self.speed

17.         if keys[pygame.K\_RIGHT] and self.rect.right < 800:

18.             self.rect.x += self.speed

19.

20.     def shoot(self, projectiles):

21.         """Create a projectile and add it to the projectiles group."""

22.         projectile = Projectile(self.rect.centerx, self.rect.top)

23.         projectiles.add(projectile)

24.

**Enemy.py**

1. import pygame

2. import random

3.

4. # Base Enemy class (abstract)

5. class Enemy(pygame.sprite.Sprite):

6.     def \_\_init\_\_(self, x, y, image):

7.         super().\_\_init\_\_()

8.         self.image = image

9.         self.rect = self.image.get\_rect()

10.         self.rect.center = (x, y)

11.         self.speed = 2

12.

13.     def update(self, \*args, \*\*kwargs):

14.         """Update enemy position (must be overridden by subclasses)."""

15.         raise NotImplementedError("Subclasses must implement update()")

16.

17. # Alien subclass (inherits from Enemy)

18. class Alien(Enemy):

19.     def \_\_init\_\_(self, x, y):

20.         image = pygame.Surface((40, 40))

21.         image.fill((255, 0, 0))  # Red square for alien

22.         super().\_\_init\_\_(x, y, image)

23.

24.     def update(self, \*args, \*\*kwargs):

25.         """Move the alien down the screen and wrap around."""

26.         self.rect.y += self.speed

27.         if self.rect.top > 600:  # Screen height

28.             self.rect.bottom = 0

29.             self.rect.x = random.randint(0, 800)  # Random x position

30.

**Projectile.py**

1. import pygame

2.

3. class Projectile(pygame.sprite.Sprite):

4.     def \_\_init\_\_(self, x, y):

5.         super().\_\_init\_\_()

6.         self.image = pygame.Surface((10, 10))

7.         self.image.fill((0, 0, 255))  # Blue square for projectile

8.         self.rect = self.image.get\_rect()

9.         self.rect.center = (x, y)

10.         self.speed = 5

11.

12.     def update(self, \*args, \*\*kwargs):

13.         """Move the projectile upward."""

14.         self.rect.y -= self.speed

15.         if self.rect.bottom < 0:  # Off-screen

16.             self.kill()

17.

**Main.py**

1. import pygame

2. import random

3. from prototype\_spaceship import Spaceship

4. from prototype\_enemy import Alien

5. from prototype\_projectiles import Projectile

6.

7. pygame.init()

8.

9. # Screen dimensions

10. SCREEN\_WIDTH = 800

11. SCREEN\_HEIGHT = 600

12.

13. # Colours

14. WHITE = (255, 255, 255)

15. BLACK = (0, 0, 0)

16.

17. # Initialise screen

18. screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

19. pygame.display.set\_caption("Space Shooter Prototype")

20.

21. # Clock for controlling frame rate

22. clock = pygame.time.Clock()

23. FPS = 60

24.

25. # Sprite groups

26. all\_sprites = pygame.sprite.Group()

27. enemies = pygame.sprite.Group()

28. projectiles = pygame.sprite.Group()

29.

30. # Create player

31. player = Spaceship()

32. all\_sprites.add(player)

33.

34. # Spawn initial enemies

35. for \_ in range(5):

36.     alien = Alien(random.randint(0, SCREEN\_WIDTH), random.randint(-100, -40))

37.     all\_sprites.add(alien)

38.     enemies.add(alien)

39.

40. # Game variables

41. score = 0

42. lives = 3

43.

44. # Timer for spawning enemy waves

45. SPAWN\_WAVE\_EVENT = pygame.USEREVENT + 1  # Custom event for spawning waves

46. pygame.time.set\_timer(SPAWN\_WAVE\_EVENT, 5000)  # Spawn a wave every 5 seconds

47.

48. # Main game loop

49. running = True

50. while running:

51.     # Handle events

52.     for event in pygame.event.get():

53.         if event.type == pygame.QUIT:

54.             running = False

55.         elif event.type == pygame.KEYDOWN:

56.             if event.key == pygame.K\_SPACE:

57.                 player.shoot(projectiles)

58.                 all\_sprites.add(projectiles)

59.         elif event.type == SPAWN\_WAVE\_EVENT:  # Spawn a new wave

60.             for \_ in range(5):  # Spawn 5 new aliens

61.                 alien = Alien(random.randint(0, SCREEN\_WIDTH), random.randint(-100, -40))

62.                 all\_sprites.add(alien)

63.                 enemies.add(alien)

64.

65.     # Update game state

66.     keys = pygame.key.get\_pressed()

67.     all\_sprites.update(keys)

68.

69.     # Check for collisions

70.     for projectile in projectiles:

71.         enemies\_hit = pygame.sprite.spritecollide(projectile, enemies, True)

72.         for enemy in enemies\_hit:

73.             projectile.kill()

74.             score += 10

75.

76.     if pygame.sprite.spritecollide(player, enemies, True):

77.         lives -= 1

78.         if lives <= 0:

79.             running = False

80.

81.     # Render

82.     screen.fill(BLACK)

83.     all\_sprites.draw(screen)

84.

85.     # Draw score and lives

86.     font = pygame.font.SysFont(None, 35)

87.     score\_text = font.render(f"Score: {score}", True, WHITE)

88.     lives\_text = font.render(f"Lives: {lives}", True, WHITE)

89.     screen.blit(score\_text, (10, 10))

90.     screen.blit(lives\_text, (10, 50))

91.

92.     # Update display

93.     pygame.display.flip()

94.     clock.tick(FPS)

95.

96. # Quit Pygame

97. pygame.quit()

98.

**Inheritance**

* The Enemy class is a **base class**that provides common functionality (e.g., image, rect, speed)
* The Alien class **inherits** from Enemy and overrides the update() method to define specific behaviour of itself (avoids code duplication)

**Encapsulation**

* Spaceship encapsulates movement (hides the details of how movement is calculated (checking boundaries, updating the rect position)) and shooting logic (hides how projectiles are created and added to the game)
* How?: Through different methods 🡪 the update() method handles movement based on key presses; the shoot() method creates a new projectile and adds it to the game
* Enemy class encapsulates common behaviour for all enemies (e.g., health, speed, position)
* Alien class encapsulates specific behaviour for aliens only
* How?: The Enemy class provides a common interface for all enemies, but the specific behaviour is hidden in subclasses
* Projectile encapsulates projectile movement
* How?: hides inside method 🡪 the update() method handles the movement of the projectile and checks if it goes off-screen

**Abstraction**

* The Enemy class is **abstract** because it defines a method (update()) that must be implemented by subclasses (hides the implementation details of specific enemies)

**Polymorphism**

* The update() method in Enemy is **overridden** in the Alien subclass 🡪 allows different enemy types to have unique behaviours while sharing a common interface (while still be treated as an object from Enemy class)
* How?: 🡪 The Enemy class defines the update() method, but it doesn’t provide an implementation. Instead, it raises an error to enforce that subclasses must override this method.

1. def update(self, \*args, \*\*kwargs):

2. """Update enemy position (must be overridden by subclasses)."""

3. raise NotImplementedError("Subclasses must implement update()")

4.

1. Testing Evidence

**Testing\_boundaries.py:**

1. import unittest

2. import pygame

3. from prototype\_spaceship import Spaceship

4. from prototype\_enemy import Alien

5. from prototype\_projectiles import Projectile

6.

7. # Initialiae Pygame

8. pygame.init()

9.

10. # Screen dimensions for testing

11. SCREEN\_WIDTH = 800

12. SCREEN\_HEIGHT = 600

13.

14. class TestBoundaries(unittest.TestCase):

15.     def setUp(self):

16.         """Set up test environment"""

17.         self.player = Spaceship(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 50, speed=5, image=pygame.Surface((50, 50)))

18.         self.alien = Alien(SCREEN\_WIDTH // 2, 0, image=pygame.Surface((40, 40)))

19.         self.projectile = Projectile(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2, image=pygame.Surface((10, 10)))

20.

21.     def test\_player\_left\_boundary(self):

22.         """Test that the player cannot move outside the left edge of the screen"""

23.         # Move player to the left edge

24.         self.player.rect.left = 0

25.         # Try to move further left

26.         self.player.update({pygame.K\_LEFT: True})

27.         # Check that the player's position hasn't changed

28.         self.assertEqual(self.player.rect.left, 0)

29.

30.     def test\_player\_right\_boundary(self):

31.         """Test that the player cannot move outside the right edge of the screen"""

32.         # Move player to the right edge

33.         self.player.rect.right = SCREEN\_WIDTH

34.         # Try to move further right

35.         self.player.update({pygame.K\_RIGHT: True})

36.         # Check that the player's position hasn't changed

37.         self.assertEqual(self.player.rect.right, SCREEN\_WIDTH)

38.

39.     def test\_enemy\_wrap\_around(self):

40.         """Test that enemies wrap around the screen when they move off the bottom edge"""

41.         # Move alien off the bottom edge

42.         self.alien.rect.top = SCREEN\_HEIGHT + 10

43.         # Update the alien's position

44.         self.alien.update()

45.         # Check that the alien has wrapped around to the top

46.         self.assertEqual(self.alien.rect.bottom, 0)

47.

48.     def test\_projectile\_off\_screen(self):

49.         """Test that projectiles are removed when they move off the top edge of the screen"""

50.         # Move projectile off the top edge

51.         self.projectile.rect.bottom = -10

52.         # Update the projectile's position

53.         self.projectile.update()

54.         # Check that the projectile is no longer alive

55.         self.assertFalse(self.projectile.alive())

56.

57. if \_\_name\_\_ == "\_\_main\_\_":

58.     unittest.main()

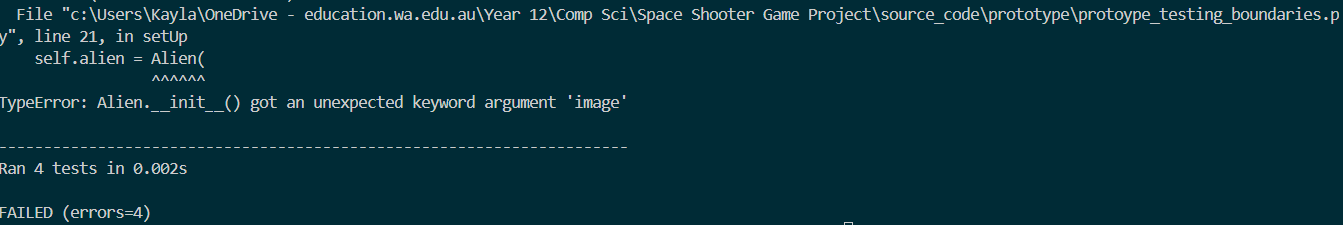
59.

First Attempt: FAILED

A screenshot of a computer

AI-generated content may be incorrect.

Second Attempt: FAILED



Third Attempt: FAILED…

A computer code on a dark background

AI-generated content may be incorrect.

Fourth Attempt: still FAILED…

A screen shot of a computer

AI-generated content may be incorrect.

Fifth Attempt: OK!!!!!

A screen shot of a computer

AI-generated content may be incorrect.

Final:

1. import unittest

2. import pygame

3. from prototype\_spaceship import Spaceship

4. from prototype\_enemy import Alien

5. from prototype\_projectiles import Projectile

6.

7. # Initialise Pygame

8. pygame.init()

9.

10. # Screen dimensions for testing

11. SCREEN\_WIDTH = 800

12. SCREEN\_HEIGHT = 600

13.

14. class TestBoundaries(unittest.TestCase):

15. def setUp(self):

16. """Set up test environment"""

17. # Create a dummy image for testing

18. self.dummy\_image = pygame.Surface((50, 50))

19.

20. # Initialise player, alien, and projectile objects

21. # Mock the Spaceship class to match its current implementation

22. self.player = Spaceship()

23. self.player.image = self.dummy\_image

24. self.player.rect = self.player.image.get\_rect()

25. self.player.rect.center = (SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 50)

26. self.player.speed = 5 # Add speed attribute for testing

27.

28. # Mock the Alien class to match its current implementation

29. self.alien = Alien(SCREEN\_WIDTH // 2, 0)

30. self.alien.image = self.dummy\_image

31. self.alien.rect = self.alien.image.get\_rect()

32. self.alien.rect.center = (SCREEN\_WIDTH // 2, 0)

33. self.alien.speed = 2 # Add speed attribute for testing

34.

35. # Mock the Projectile class to match its current implementation

36. self.projectile = Projectile(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2)

37. self.projectile.image = self.dummy\_image

38. self.projectile.rect = self.projectile.image.get\_rect()

39. self.projectile.rect.center = (SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2)

40. self.projectile.speed = 5 # Add speed attribute for testing

41.

42. def test\_player\_left\_boundary(self):

43. """Test that the player cannot move outside the left edge of the screen"""

44. # Move player to the left edge

45. self.player.rect.left = 0

46. # Simulate the left arrow key being pressed

47. keys = {pygame.K\_LEFT: True, pygame.K\_RIGHT: False}

48. # Try to move further left

49. self.player.update(keys)

50. # Check that the player's position hasn't changed

51. self.assertEqual(self.player.rect.left, 0)

52.

53. def test\_player\_right\_boundary(self):

54. """Test that the player cannot move outside the right edge of the screen"""

55. # Move player to the right edge

56. self.player.rect.right = SCREEN\_WIDTH

57. # Simulate the right arrow key being pressed

58. keys = {pygame.K\_LEFT: False, pygame.K\_RIGHT: True}

59. # Try to move further right

60. self.player.update(keys)

61. # Check that the player's position hasn't changed

62. self.assertEqual(self.player.rect.right, SCREEN\_WIDTH)

63.

64. def test\_enemy\_wrap\_around(self):

65. """Test that enemies wrap around the screen when they move off the bottom edge"""

66. # Move alien off the bottom edge

67. self.alien.rect.top = SCREEN\_HEIGHT + 10

68. # Update the alien's position

69. self.alien.update()

70. # Check that the alien has wrapped around to the top

71. self.assertEqual(self.alien.rect.bottom, 0)

72.

73. def test\_projectile\_off\_screen(self):

74. """Test that projectiles are removed when they move off the top edge of the screen"""

75. # Move projectile off the top edge

76. self.projectile.rect.bottom = -10

77. # Update the projectile's position

78. self.projectile.update()

79. # Check that the projectile is no longer alive

80. self.assertFalse(self.projectile.alive())

81.

82. if \_\_name\_\_ == "\_\_main\_\_":

83. unittest.main()

84.

1. Prepare Documentation

In structuring my classes, I aimed to follow core principles of OOP, focusing on clear organisation and separation of concerns. For example, I created a **Spaceship** class to handle the player’s movement and shooting logic, keeping these functionalities encapsulated and simple. The **Enemy** class acts as a base class, allowing me to easily add different types of enemies, such as the **Alien** class, which inherits and overrides the update() method to define unique movement behaviour. This approach reduces code duplication and allows for easy extension in the future. I used inheritance to create common behaviour across different enemy types while implementing polymorphism to ensure each subclass has its own specific implementation. For collision detection and projectile management, I encapsulated the relevant logic inside their own classes, making the code modular and easier to maintain. Overall, the design focuses on reusability, simplicity, and flexibility while keeping each class responsible for one core functionality.