File initialization

Library use:

1. Pygame
2. Random
3. Math
4. Sys: provides system-specific functionality (might be used for program termination)
5. Pygame.font
6. Pygame.mixer: provides audio functionality.

1. **Initialization Constants**:
   - **pygame.init():** Initializes Pygame, essential for using its features. This step sets up subsystems like audio, video, and event handling.
   - **WIDTH, HEIGHT = 800, 600:** These constants define the dimensions of the game window (800 pixels wide and 600 pixels tall).
   - **win = pygame.display.set_mode((WIDTH, HEIGHT)):** This creates the game window with the specified dimensions.
   - **pygame.display.set_caption("SMASH IT"):** This sets the window title to "SMASH IT".

2. **Game Settings**:
   - **FPS = 60**: This sets the frames per second for the game loop.
   - **BALL_RADIUS = 10**: This specifies the radius of the game ball.
   - **lives = 3:** This initializes the player's lives to 3.

Classes:

1. **Class Name**: Ball
2. **Attributes**:
   - x: The x-coordinate of the ball's center.
   - y: The y-coordinate of the ball's center.
   - radius: The radius of the ball.
   - color: The color of the ball (e.g., RGB value or predefined color name).
   - x_vel: The horizontal velocity of the ball (initialized to 0).
   - y_vel: The vertical velocity of the ball (initialized to a negative value, indicating upward movement).
3. **Methods**:
   - __init__(self, x, y, radius, color): The constructor method initializes the ball object with the specified position, radius, and color. It also sets the initial velocities to zero (horizontal) and upward (vertical).

   - move(self): This method updates the ball's position based on its current velocities. It adds x_vel to x and y_vel to y.

   - set_vel(self, x_vel, y_vel): This method allows you to set the ball's velocities (both horizontal and vertical) to specific values.

   - draw(self, win): This method draws the ball on the game window (win). It uses pygame.draw.circle() to create a circle at the specified (x, y) position with the given radius and color.

Class Name: Paddle

1. **Attributes**:
   - x: The x-coordinate of the paddle's top-left corner.
   - y: The y-coordinate of the paddle's top-left corner.
   - width: The width of the paddle (initialized to the value of Paddle.width_paddle).
   - height: The height of the paddle (initialized to the value of Paddle.height_paddle).
   - color: The color of the paddle (e.g., RGB value or predefined color name).
   - vel: The velocity of the paddle (initialized to 4).

2. **Methods**:
   - __init__(self, x, y, width, height, color): The constructor method initializes the paddle object with the specified position, dimensions, and color.

   - draw(self, win): This method draws the paddle on the game window (win). It uses pygame.draw.rect() to create a rectangle at the specified (x, y) position with the given width, height, and color.

   - inc_Vel(self): This method increases the paddle's velocity by 4 units.

   - move(self, direction=5): This method moves the paddle horizontally by adjusting its x position based on the current velocity (vel) and the specified direction.

   - inc_paddle(self): This method increases the paddle's width by 50 units.

Class Name: Brick

1. **Attributes**:
   - x: The x-coordinate of the brick's top-left corner.
   - y: The y-coordinate of the brick's top-left corner.
   - width: The width of the brick.
   - height: The height of the brick.
   - health: The current health (strength) of the brick.
   - max_health: The maximum health (initial health) of the brick.
   - colors: A list of colors representing the brick's appearance (e.g., gradient colors).
   - color: The current color of the brick (initialized to the first color in the colors list).
2. **Methods**:
   - __init__(self, x, y, width, height, health, colors): The constructor method initializes the brick object with the specified position, dimensions, health, and colors.

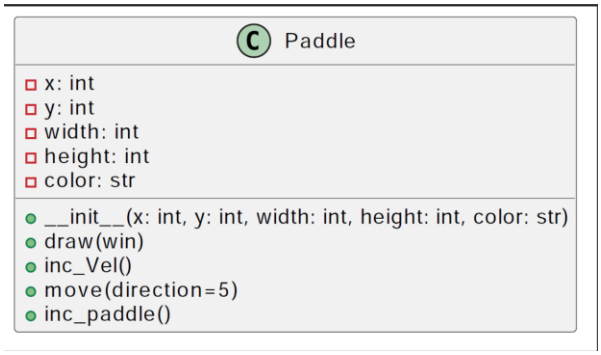
   - draw(self, win): This method draws the brick on the game window (win). It uses pygame.draw.rect() to create a rectangle at the specified (x, y) position with the given width, height, and current color.

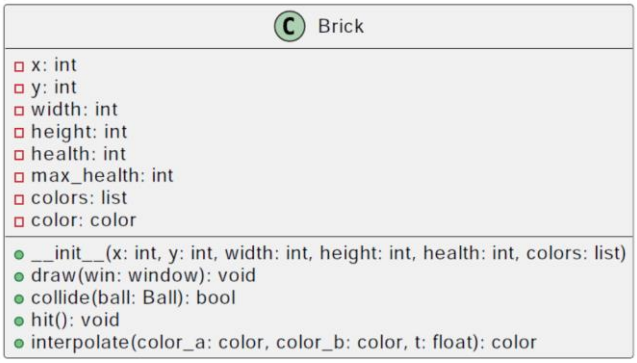
   - collide(self, ball): This method checks if the ball collides with the brick. If so, it reduces the brick's health, updates the ball's velocity, and returns True. Otherwise, it returns False.

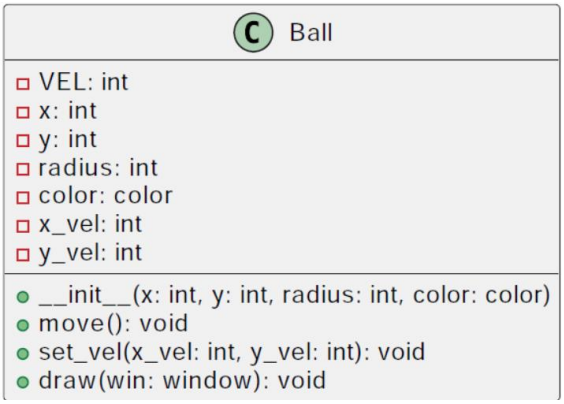
   - hit(self): This method reduces the brick's health by 1 and updates its color based on the remaining health.

- interpolate(color_a, color_b, t): This static method interpolates between two colors (color_a and color_b) based on a parameter t (ranging from 0 to 1). It returns a new color that represents a blend between the two colors.

UML classes:







FUNCTIONS IN FUNCTION FILE: 10 FUNCTIONS

Function Name: resize_image(image)

- **Purpose**: This function resizes an input image to a smaller size (30x30 pixels).
- **Parameters**:
  - image: The input image (presumably loaded using Pygame).
- **Return Value**: The resized image.
- **Explanation**:
  - The function uses pygame.transform.scale() to resize the input image.
  - It scales the image to a new size of 30x30 pixels.
  - The resized image is returned as the result.

Function Name: play_song(filepath)

- **Purpose**: This function plays a music file (presumably in the background during gameplay).
- **Parameters**:
  - filepath: The path to the music file (e.g., MP3, WAV, etc.).
- **Explanation**:
  - The function loads the specified music file using pygame.mixer.music.load(filepath).
  - It then plays the loaded music indefinitely (-1 indicates continuous looping).
  - The music will continue playing until explicitly stopped or paused.

Function Name: show_start_screen()

1. **Purpose**:
   - This function displays the start screen for your game.
   - It includes instructions for the player, such as how to play, mute/unmute music, and quit the game.
   - The start screen also shows the game title and background image.
2. **Steps**:
   - Load the background image from the file "./img/start_screen.jpg".
   - Handle user input events:
     - If the user presses ‘M’, mute the music.
     - If the user presses ‘N’, unmute the music.
     - If the user presses ‘Q’, quit the game.
     - Otherwise, start the game when any other key is pressed.

Function Name: generate_bricks(rows, cols)

- **Purpose**:
  - This function generates a list of brick objects for the game.
  - The bricks are arranged in a grid with the specified number of rows and columns.
  - Each brick has a specific position, width, height, health, and color.
- **Parameters**:
  - rows: The number of rows of bricks.
  - cols: The number of columns of bricks.
- **Return Value**: A list of Brick objects representing the bricks.
- **Explanation**:
  - Calculate the brick_width based on the game window width (WIDTH) and the number of columns.
  - Set the brick_height (adjust as needed).
  - Define the padding_top (adjust as needed) to create space at the top.
  - Initialize a variable l to track the level (number of shots needed to break a brick).
  - Create an empty list called bricks.
  - Loop through each row and column:
  - Calculate the brick_y position based on the padding, brick height, and row index.
  - Create a Brick object with the specified position, dimensions, level, and color.
  - Append the brick to the bricks list.
  - Increment the level (l) for the next row.
  - Return the list of bricks.

Function Name: draw(win, paddle, ball, bricks, lives)

- **Purpose**:
  - This function draws the game elements on the screen.
  - It includes the background image, paddle, ball, bricks, and heart/lives images.
- **Parameters**:
  - win: The game window surface.
  - paddle: The player-controlled paddle object.
  - ball: The game ball object.
  - bricks: A list of Brick objects representing the bricks.
  - lives: The number of remaining lives.
- **Explanation**:
  - Load the background image from the file "./img/space3.jpg" and display it at position (0, 0).
  - Draw the paddle and ball using their respective draw() methods.
  - Loop through each brick in the bricks list and draw it using the draw() method.
  - Display heart/lives images in a row:
  - Calculate the position for each heart image based on the desired size (new_width and new_height).
  - Load the appropriate heart image based on the remaining lives.
  - Resize the image to the desired size.
  - Adjust the Y position as needed.
  - Update the display to show all the drawn elements.

Function Name: draw_heart(win, center_x, center_y, width, height, color)

- **Purpose**:
  - This function draws a heart shape on the game window.
  - The heart shape is defined by a set of points.
- **Parameters**:
  - win: The game window surface.
  - center_x: The x-coordinate of the heart's center.
  - center_y: The y-coordinate of the heart's center.
  - width: The width of the heart.
  - height: The height of the heart.
  - color: The color of the heart (e.g., RGB value or predefined color name).
- **Explanation**:
  - Define a list of points that form the heart shape:
  - (center_x, center_y + height // 3): Top point.
  - (center_x - width // 2, center_y): Left point.
  - (center_x, center_y - height // 3): Bottom point.
  - (center_x + width // 2, center_y): Right point.
  - (center_x, center_y + height // 3): Top point (closing the shape).
  - (center_x - width // 4, center_y + height * 2 // 3): Left bottom point.
  - (center_x + width // 4, center_y + height * 2 // 3): Right bottom point.
  - Use pygame.draw.polygon() to draw the heart shape on the window with the specified color.

Function Name: ball_collision(ball)

- **Purpose**:
  - This function handles collisions of the game ball with the screen boundaries.
  - If the ball hits the left or right boundary, it reflects horizontally.
  - If the ball hits the top or bottom boundary, it reflects vertically.
- **Parameters**:
  - ball: The game ball object.
- **Explanation**:
  - Check if the ball's left edge (ball.x - BALL_RADIUS) is outside the left boundary (0) or the right edge (ball.x + BALL_RADIUS) is outside the right boundary (WIDTH).
  - If so, reverse the horizontal velocity (ball.x_vel) to make the ball bounce off the boundary.
  - Check if the ball's top edge (ball.y - BALL_RADIUS) is outside the top boundary (0) or the bottom edge (ball.y + BALL_RADIUS) is outside the bottom boundary (HEIGHT).
  - If so, reverse the vertical velocity (ball.y_vel) to make the ball bounce off the boundary.

Function Name: ball_paddle_collision(ball, paddle)

- **Purpose**:
  - This function handles collisions between the game ball and the player-controlled paddle.
  - It calculates the new velocity for the ball after hitting the paddle based on the position of the collision.
- **Parameters**:
  - ball: The game ball object.
  - paddle: The player-controlled paddle object.
- **Explanation**:
  - Check if the ball's x-coordinate is within the paddle's x-range (paddle.x to paddle.x + paddle.width).
  - If not, there is no collision, so return.
  - Check if the ball's y-coordinate (bottom edge) is below the paddle's top edge (paddle.y).
  - If not, there is no collision, so return.
  - Calculate the distance from the ball's center to the paddle's center (distance_to_center).
  - Convert this distance to a percentage of the paddle's width (percent_width).
  - Calculate the angle of reflection based on this percentage.
  - Convert the angle to radians (angle_radians).
  - Compute the new x-velocity (x_vel) and y-velocity (y_vel) for the ball based on the angle.
  - Set the ball's velocities to the new values.

Function Name: win_screen()

- **Purpose**:
  - This function displays a win screen when the player successfully completes the game.
  - It shows an image (loaded from "img/win.png").
  - The player can press the 'Q' key to quit the game.
- **Explanation**:
  - Load the win screen background image from the file "img/win.png".
  - Display the background image at position (0, 0) on the game window.
  - Update the display to show the win screen.
  - Wait for user input:
  - If the user presses the 'Q' key, exit the game by quitting Pygame and calling quit().

Function Name: lose_screen()

- **Purpose**:
  - This function displays a lose screen when the player loses the game.
  - It shows an image (loaded from "./img/you_lost.jpg").
  - The player can press the 'Q' key to quit the game.
- **Explanation**:
  - Load the lose screen background image from the file "./img/you_lost.jpg".
  - Display the background image at position (0, 0) on the game window.
  - Set up a font (using pygame.font.SysFont(None, 58)).
  - Render the text "Press Q to quit" in white color.
  - Display the rendered text at an appropriate position.
  - Update the display to show the lose screen.
  - Wait for user input:
  - If the user presses the 'Q' key, exit the game by quitting Pygame and calling quit().

MAIN FILE:

1. **Initialization and Setup for items (potions)**:
   - x = randint(0, 500): This initializes the variable x with a random integer between 0 (inclusive) and 500 (exclusive).
   - y = 100: This initializes the variable y with the value 100.
   - speed = 2: This initializes the variable speed with the value 2.
   - radius = 50: This initializes the variable radius with the value 50.
   - random_number = 0: This initializes the variable random_number with the value 0.
   - Loading images for different potions:
   - potion_live: Loads an image from the file "img/lives.png".
   - potion_flash: Loads an image from the file "img/flash.png".
   - potion_paddle: Loads an image from the file "img/green.png".

2. **Functions for Items (Potions)**:
   - incLives(): Increases the global lives variable by 1 (for adding extra lives).
   - init(): Initializes the position of items (potions). If y exceeds the screen height, it resets y and randomizes x

3. **Game Loop and Display Setup**:
   - clock = pygame.time.Clock(): Creates a clock object for controlling the frame rate.
   - display_output = [800, 600]: Sets the display size to 800x600 pixels.
   - screen = pygame.display.set_mode(display_output): Creates the game window with the specified dimensions.

4. Function Name: main()

- **Purpose**:
  - This function represents the main game loop and setup.
  - It initializes various game elements such as the clock, paddle, ball, bricks, and starts playing background music.
  - The game starts by showing the start screen (main menu).
- **Explanation**:
  - Initialize the game clock using pygame.time.Clock().
  - Play the background music from the file "sounds/song.mp3" using play_song("sounds/song.mp3").
  - Calculate the initial position for the player-controlled paddle (paddle_x and paddle_y).
  - Create a Paddle object with the specified position, dimensions, and color.
  - Create a Ball object with the specified position, radius, and color.
  - Set the number of rows and columns for the bricks (rows and columns).
  - Generate the bricks using the generate_bricks(rows, columns) function.
  - Show the start screen (main menu) using show_start_screen().

5. Function Name: radom_num()

- **Purpose**:
  - This function generates a random number for items (potions).
  - Depending on the random number, it performs different actions (e.g., increase lives, increase paddle width, or increase paddle speed).
- **Explanation**:
  - Generate a random integer between 1 and 4
  - Based on the value of random_number, perform the corresponding action:
  - If random_number is 1, increase the player's lives.
  - If random_number is 2, increase the paddle width.
  - If random_number is 3, increase the paddle speed.

6. Function Name: items()

- **Purpose**:
  - This function draws the items (potions) on the screen.
  - The items move downward (y increases by speed).
  - Depending on the value of random_number, it blits the appropriate potion image at the specified position.
- **Explanation**:
  - Update the y position of the items (potions) by adding the speed.
  - Depending on the value of random_number, blit the corresponding potion image (potion_live, potion_paddle, or potion_flash) at the position (x, y).
  - Update the display to show the drawn item

Game Loop:

- The game loop runs while the **run** variable is True.
- The clock is ticked at a specified frame rate (FPS).
- If the background music is not playing (pygame.mixer.music.get_busy() returns False), it starts playing the song from the file "sounds/song.mp3".
- The game loop handles events (such as quitting the game) using pygame.event.get().
- The player can move the paddle left (pygame.K_LEFT) or right (pygame.K_RIGHT) using the arrow keys.
- The ball's position is updated using ball.move().
- Collisions between the ball and screen boundaries are checked using ball_collision(ball).
- Collisions between the ball and the paddle are checked using ball_paddle_collision(ball, paddle).
- Bricks are iterated through, and collisions with the ball are detected using brick.collide(ball).
- If a brick's health reaches 0, an item (potion) is generated using radom_num(), and the brick is removed from the list.
- If the ball falls below the screen, the player loses a life, and the ball is reset.
- If there are no lives left, the lose screen is displayed using lose_screen().
- If all bricks are destroyed, the win screen is displayed using win_screen().
- The items are drawn using items().
- The game elements (paddle, ball, bricks, lives) are drawn using draw(win, paddle, ball, bricks, lives).
- The game loop continues until the user quits the game.

Main Execution:

- The main() function is called when the script is executed.
- The game setup and loop are handled within the main() function.