CS 3354.003
HW #4
Jay Shingari JXS200089

PART D:

Person Class

1. Initialization (Person() constructor):
   - Initializes images array with `GreenfootImage objects.
   - Sets the initial image for the Person.

2. Action (act() method):
   - Applies gravity by adjusting the Person's vertical position.
   - Checks if the space key is pressed and if the `Person` is not already jumping, then calls jump().
   - Calls animate() to cycle through images for animation.

3. Gravity (applyGravity() method):
   - Adjusts the Person's vertical position based on velocity.
   - Checks if the Person has landed (reached a certain Y position) and resets the jumping status and velocity if so.

4. Jumping (jump() method):
   - Sets the velocity to a negative value to simulate upward motion.
   - Sets isJumping to true indicating that the Person is in the process of jumping.

5. Animation (animate() method):
   - Cycles through the images array to create an animation effect.

Obstacle Class

1. Action (act() method):
   - Moves the Obstacle at a specified speed.
   - Resets the Obstacle position when it reaches the edge of the world.
   - Checks for collision with Person and stops the game if a collision occurs.

In the Greenfoot game environment, the Person and Obstacle classes interact through a series of background processes driven by the main game loop. The Person class, upon initialization, sets up an array of images for animation and continuously checks for user input (like the space bar for jumping) in its act() method. It simulates gravity by adjusting vertical velocity and position, and employs an animation cycle to visually represent movement and jumping. Conversely, the Obstacle class moves across the screen at a set speed, resetting its position upon reaching the world's edge and continuously checking for collisions with the Person. If a

collision is detected, it halts the game. These interactions are orchestrated by Greenfoot's game loop, which repetitively calls the act() method of both classes, ensuring smooth animation, responsive controls, and the dynamic handling of game logic like collision detection and movement, creating an engaging and interactive game experience.