PyQt5 Advanced Exam: Multithreading and Animation

Exercise 1: Implement a Worker Thread Using QThread (5 points)

Create a PyQt5 application that uses `QThread` to perform a time-consuming task (e.g., counting from 1 to 100) in the background without freezing the main GUI. Display the progress in a `QProgressBar`.

Requirements:

- Use a `QThread` to handle the counting task.
- Emit progress updates from the worker thread to the main thread using signals.
- Update the `QProgressBar` in real-time.

—

Exercise 2: Create a Countdown Timer Using QTimer (5 points)

Build a PyQt5 application that displays a countdown timer starting from 10 seconds. Use `QTimer` to update the countdown every second. When the timer reaches 0, display a message box saying "Time's up!".

Requirements:

- Use `QTimer` to handle the timing.
- Display the countdown in a `QLabel`.
- Use `QMessageBox` to show the "Time's up!" message.

—

Exercise 3: Create an Animated Object Using QPropertyAnimation (5 points)

Create a PyQt5 application that animates a `QPushButton` moving from the top-left corner of the window to the bottom-right corner over 5 seconds. Use `QPropertyAnimation` to handle the animation.

Requirements:

- Use `QPropertyAnimation` to animate the `pos` property of the button.
- Set the animation duration to 5000 milliseconds (5 seconds).

—

Exercise 4: Build a Custom Graphics Scene Using QGraphicsScene (5 points)

Create a PyQt5 application that uses `QGraphicsScene` and `QGraphicsView` to display a custom scene. The scene should include:

- A rectangle (`QGraphicsRectItem`) at the center of the scene.
- A circle (`QGraphicsEllipseItem`) that moves along the edges of the rectangle in a continuous loop.
- Use `QTimer` to animate the circle's movement.

Requirements:

- Use `QGraphicsScene` and `QGraphicsView` to render the scene.
- Animate the circle's movement using `QTimer`.
- Ensure the circle follows the rectangle's edges smoothly.

—