Understanding the Game Loop with Timing Control - Simple

19MAY24

Feature/#12/fps&ups #13

Adam Pinnock

Introduction

This guide explains how a game loop with timing control works, focusing on the snippet provided. It will help you understand the role of different clocks, timing, and how the loop maintains a consistent update and render rate.

Overview of the Game Loop

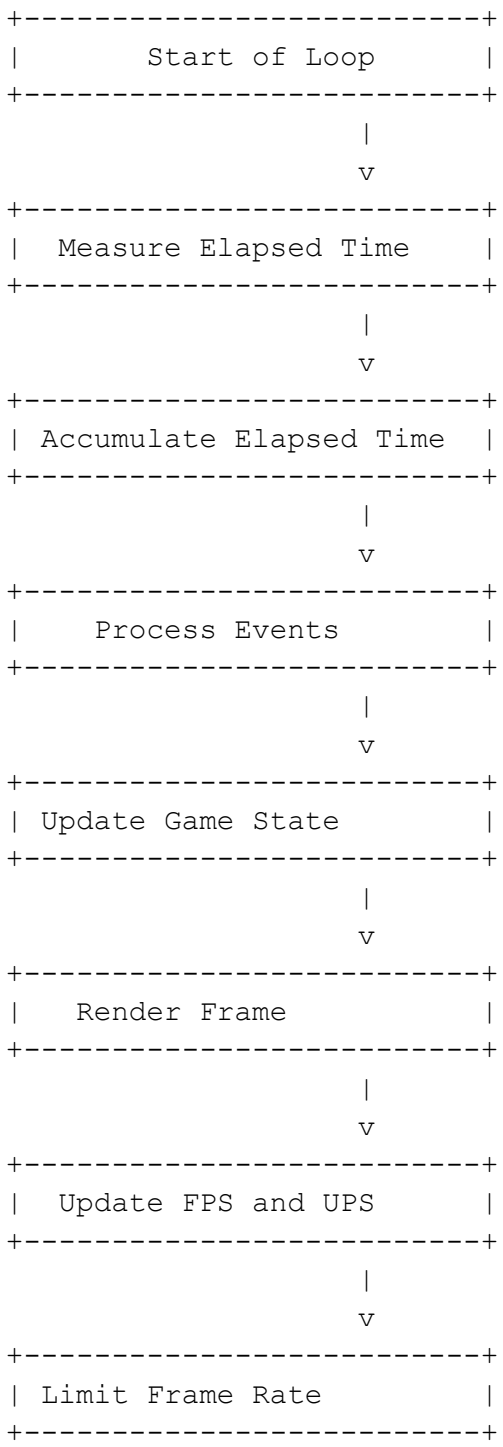
The game loop performs the following actions in each iteration:

1. Measures elapsed time.
2. Processes events (like user inputs).
3. Updates the game state.
4. Renders the game.
5. Sleeps to maintain a consistent frame rate.

Key Variables and Constants

• elapsedTime: The time elapsed since the last frame.
• timeSinceLastUpdate: Accumulates time to determine when to perform the next update.
• timeSinceLastRender: Accumulates time to determine when to render the next frame.
• fpsUpdateTime: Tracks time to update the FPS and UPS display.
• TimePerUpdate: The fixed time interval for updates (e.g., 1/30th of a second for 30 UPS).
• TimePerFrame: The fixed time interval for rendering frames (e.g., 1/60th of a second for 60 FPS).

Flowchart of the Loop



Detailed Explanation

1. Measuring Elapsed Time

At the beginning of each loop iteration, the clock is restarted to measure the time elapsed since the last frame.

```
sf::Time elapsedTime = clock.restart();
```

2. Accumulating Elapsed Time

The elapsed time is added to timeSinceLastUpdate, timeSinceLastRender, and fpsUpdateTime to keep track of time for updates, rendering, and FPS/UPS updates.

```
timeSinceLastUpdate += elapsedTime;
timeSinceLastRender += elapsedTime;
fpsUpdateTime += elapsedTime;
```

3. Processing Events

Events (like keyboard and mouse inputs) are processed to keep the application responsive.

```
processEvents();
```

4. Updating Game State

The game state is updated in fixed intervals to ensure consistent gameplay mechanics, independent of frame rate.

```
while (timeSinceLastUpdate >= TimePerUpdate) {
    timeSinceLastUpdate -= TimePerUpdate;
    update(TimePerUpdate);
    updates++;
    logUpdate(i);
}
```

5. Rendering Frame

Rendering happens at a fixed frame rate. If sufficient time has passed since the last render, the game frame is rendered.

```
if (timeSinceLastRender >= TimePerFrame) {
    render();
    frames++;
    timeSinceLastRender -= TimePerFrame;
}
```

6. Updating FPS and UPS

FPS and UPS are updated and displayed once per second to monitor performance.

```
if (fpsUpdateTime >= sf::seconds(1.0f)) {
    updateFPSandUPS(fpsUpdateTime);
    fpsUpdateTime -= sf::seconds(1.0f);
}
```

7. Limiting Frame Rate

The remaining time for the current frame is calculated and the program sleeps to maintain a consistent frame rate.

```
sf::Time sleepTime = TimePerFrame - clock.getElapsedTime();
if (sleepTime > sf::Time::Zero) {
    sf::sleep(sleepTime);
}
```