Game Programming Patterns _ Sequencing _ Game Loop

"Decouple the progression of game time from user input & processor speed"

* almost every game has a game loop, no two are the same & few programs outside games use them.

Batch Mode Programs: dump code in, push btn, wait, got results, done.
Interactive " : get immediate feedback, it waits for input & responds

* Event Loops still block all processing until user input is received but games now don't stop when the user stops

① game loop processes user input but doesn't wait for it.

```
while (true) {
   processInput(); \\ any user input since last call | Process User Input
   update(); \\ adv. game sim. 1 step (AI -> Physics) | Update game state
   render(); \\ draw game to show what happened | renders game
}   + tracks passage of time to control the rate of gameplay
```

- one crank of game loop is a tick or frame, then you compare w/ human time & get frames/sec. or FPS
- FPS is affected by how much changed/frame + speed of underlying platform

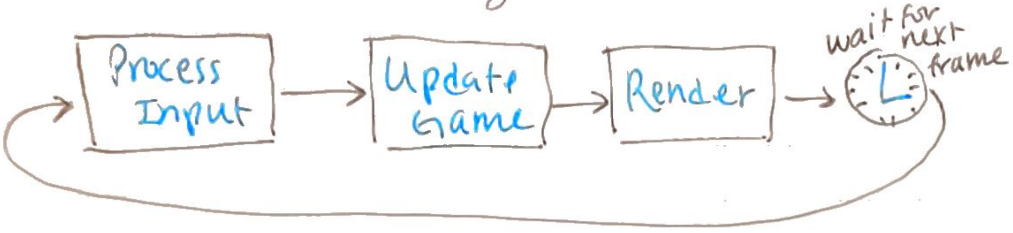
② it runs the game at a consistent speed despite underlying hardware differences.

- Library: you own main game loop + call into the library
- Engine: eng. owns the loop + calls into Your code

* Need to be careful about performance in the game loop
* if you're building on top of OS/platform w/ its own loop you'll need to make the two loops play nicely w/ each other.

Unity's gameloop   Witter article on game loops   Glenn Fiedler's Fix Your Timestep

Need to control timing

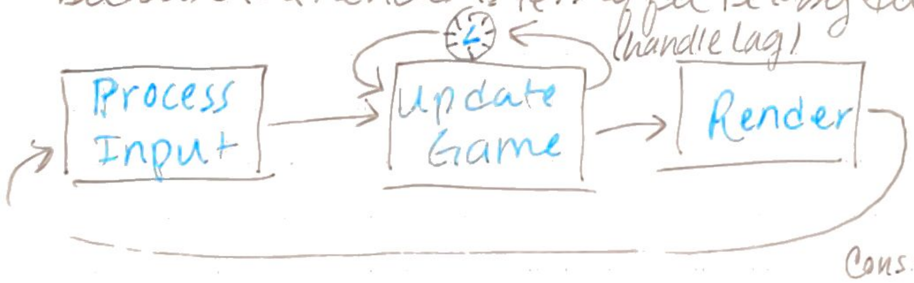


Problem:

① each update() adv. the game by a certain amt
② it takes a certain amt of real time to process that
if 2 > 1, game slows
→ Choose a timestep based on ②, ↑frame ↑steps game makes
called variable or fluid
→ w/ variable timestep you scale the velocity (of a bullet for ex.) by the elapsed time. It will travel in the same amt of real time. But now the game isn't deterministic.

* game physics engines are approx. of the real laws of physics damping is applied & tuned to a specific timestep to avoid the game physics from blowing up.
→ there can also be rounding errors w/ floating point repr. of variables that are compounded at ↑FPS.

* we can allow flexibility in rendering to free up processor time because the render is less affected by the timestep.



* Now that rendering is out of the game loop the game is simulated at a const. rate but the view (render) is choppier on a slower machine

* if a render happens b/w updates we can pass the cur lag into render() & the code can extrapolate based on velocity.
- who owns the game loop depends on the platform.
* need to be aware of power consumption.