

COMP150: Game Development Practices

The main loop

Basic game architecture

- ▶ CPUs execute **sequences of instructions**
 - ▶ At the CPU level, there is no such thing as “when X happens, do Y ”
 - ▶ Instead: “keep checking whether X has happened; if so, do Y ”
- ▶ Thus games need to have a **main loop**
- ▶ The main loop is usually part of the game's **engine**
 - ▶ Engines and high level frameworks (e.g. Kivy, Unity, Unreal) usually implement the main loop for you
 - ▶ Low level frameworks (e.g. SDL, OpenGL, DirectX) usually require you to implement your own main loop

The basic main loop

The most basic main game loop does **three** things:

1. Handle **input**

- ▶ Mouse, keyboard, joypad etc.
- ▶ Operating system events (minimise, close, alt+tab etc.)

2. **Update** the state of the game

- ▶ Physics, collision detection, AI etc.

3. **Render** the game to the screen

It does these **once per frame** (typically 30 or 60 times per second)

The basic main loop

```
bool running = true;

while (running)
{
    handleInput();
    update();
    render();
}
```

Handling input

There are two ways of handling input in a game:

- ▶ By responding to **events**

- ▶ `SDL_PollEvent`
- ▶ See https://wiki.libsdl.org/SDL_EventType for a list of event types

- ▶ By querying **state**

- ▶ `SDL_GetKeyboardState`, `SDL_GetMouseState`, `SDL_GameControllerGetAxis`, etc.

What's the difference?

- ▶ **Event**: "The space bar was (pressed / released)"
- ▶ **State**: "The space bar is (down / up) right now"

Updating the game state

- ▶ Generally this is where your **game logic** is implemented
- ▶ I.e. anything not directly related to input or graphics
- ▶ What goes in here depends on the game...

Rendering

- ▶ This is where you draw the **current state of the game** to the screen
- ▶ Also draw any **heads-up display (HUD)** elements, e.g. score, lives, mini-map, etc.
- ▶ Graphical effects (animations, particles) may be handled **either** in the render step **or** in the update step (but be consistent)
- ▶ In frameworks like SDL, you generally **redraw everything** on every frame
- ▶ Rendering in SDL is **double buffered**
 - ▶ `SDL_Render...` functions actually draw to an **off-screen buffer**
 - ▶ `SDL_RenderPresent` displays the off-screen buffer on screen

Screen refresh rate

- ▶ Old CRT monitors worked by scanning an electron beam down the screen
 - ▶ https://www.youtube.com/watch?v=lRidfW_l4vs
- ▶ Hence the term **(vertical) refresh rate**
- ▶ Refresh rate is measured in **cycles per second** i.e. **Hz**
- ▶ Other monitor technologies work differently, but still refresh the screen at regular intervals

Frame rate

- ▶ We generally want to sync it up so that
one display refresh = one main loop iteration
- ▶ If the main loop runs too slowly, we get “lag”
- ▶ If the main loop runs too quickly, we waste resources on drawing things faster than the display can show them

Limiting the frame rate

- ▶ If the renderer was created with the `SDL_RENDERER_PRESENTVSYNC` flag, `SDL_RenderPresent` waits for the next vertical blank
- ▶ This limits the game's frame rate to the refresh rate of the device
- ▶ However, refresh rates can vary
 - ▶ Older TVs: $\sim 30\text{Hz}$
 - ▶ HDTVs and standard monitors: 60Hz
 - ▶ High-end "gaming" monitors: 120Hz or higher

Limiting the update rate

- ▶ Having the update frequency depend on the refresh rate would be bad!
 - ▶ The game could appear to run in slow or fast motion, completely changing the gameplay
- ▶ This was the situation on older consoles:
American/Japanese versions of games actually ran a little faster than European versions, due to the NTSC TV standard having a higher refresh rate than PAL!

Variable time step

- ▶ Have the update step depend on the **elapsed time since the last update**
- ▶ Also known as the **delta time**
- ▶ E.g. instead of this:

```
player.positionX += player.velocityX;
```

- ▶ do this:

```
player.positionX += player.velocityX * deltaTime;
```

Measuring elapsed time

```
bool running = true;
Uint32 lastFrameTime = SDL_GetTicks();

while (running)
{
    Uint32 currentTime = SDL_GetTicks();
    Uint32 deltaTime = currentTime - lastFrameTime;
    // deltaTime is the number of milliseconds since ←
    // the last update

    handleInput();
    update(deltaTime);
    render();

    lastFrameTime = currentTime;
}
```

Variable time step

- ▶ **Good:** the game will no longer run in slow or fast motion at different refresh rates
- ▶ **Bad:** may increase the complexity of the `update` function
- ▶ **Bad:** some systems (e.g. physics) may become prone to numerical errors

Fixed time step

- ▶ Perform the update at a **fixed rate**, e.g. 60 times per second
- ▶ If refresh rate $< 60\text{Hz}$, update several times per frame
- ▶ If refresh rate $> 60\text{Hz}$, update once every few frames

Fixed time step

```
bool running = true;
Uint32 lastUpdateTime = SDL_GetTicks();
const Uint32 timePerUpdate = 1000 / 60;

while (running)
{
    Uint32 currentTime = SDL_GetTicks();
    handleInput();

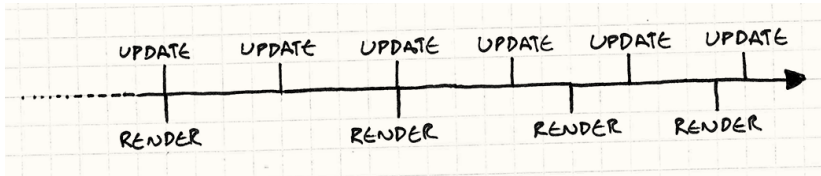
    while (currentTime - lastUpdateTime >= ←
           timePerUpdate)
    {
        update();
        lastUpdateTime += timePerUpdate;
    }

    render();
}
```


Stalling

- ▶ What if `update` takes longer than `timePerUpdate` to execute?
- ▶ The `while` loop will perform more and more iterations in an effort to catch up, eventually grinding the game to a halt
- ▶ **Solution:** `break` out of the loop after a maximum number of iterations (e.g. 10)

Interpolation

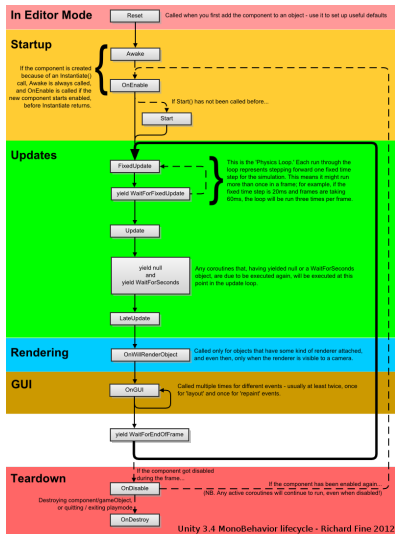


- ▶ Rendering at “irregular” intervals (with respect to update) can result in jerky movement
- ▶ Solution: **interpolate** between the two previous updates
 - ▶ E.g. if the render falls exactly halfway between two updates, render each object exactly halfway between its positions **before** and **after** the most recent update

Further information on fixed time steps

- ▶ <http://gafferongames.com/game-physics/fix-your-timestep/>
- ▶ <http://gameprogrammingpatterns.com/game-loop.html>

The “main loop” in Unity



Summary

- ▶ The **main loop** of a game runs once per frame, and handles **input**, **updating** and **rendering**
- ▶ Using a **fixed time step** is a good idea, but be careful of **stalling**