**Game development**

1. **Ensure all images elements load on the screen**

window.onload = () => {

// Function that references the image elements

}

🡪 The actions will happen in the .onload, after everything has been setup.

1. **Game loop**

const loop = () => {

update();

draw();

window.requestAnimationFrame(loop);

}

loop(); // Initiate first loop

🡪 Function loop() will be called all over again. Each call is called a “frame”. Depending on different computers, the frame times may be different. For new computers, the frame time might be **60 frames per second (60 fps)**.

1. [**Dynamic frame rates**](https://spicyyoghurt.com/tutorials/html5-javascript-game-development/create-a-smooth-canvas-animation#:~:text=Effects%20of%20dynamic%20frame%20rate%20on%20the%20animation)

Since frame rates may be different across devices, there is a need for handling dynamic frame rates, in order to compensate for devices that have low frame rates. The solution is to include **time** into our code.

🡪 We calculate the time it takes for each frame. Then we apply this **time**  variable into update functions, which accounts for the movements of objects relative to time.

“When your game runs at 60fps, that's roughly 0.0167 seconds per frame. This means that when you want to move an object with 50 pixels per second, you have to multiply 50 by the number of seconds that have passed since the last frame. A game running at 60 fps will move the object 0.835 pixels per frame. That's what's happening in the update() function.

When the frame rate increases or decreases, the movement speed will too. No matter how much time has passed, your objects will always move at the **desired speed**. This makes the animation more suitable for different kinds of hardware, with **different frame rates**. The displacement will be constant”

let secondsPassed = 0;

let oldTimeStamp = 0;

let movingSpeed = 50;

function gameLoop(timeStamp) {

// Calculate how much time has passed

secondsPassed = (timeStamp - oldTimeStamp) / 1000;

oldTimeStamp = timeStamp;

// Pass the time to the update

update(secondsPassed);

draw();

window.requestAnimationFrame(gameLoop);

}

function update(secondsPassed) {

// Use time to calculate new position

rectX += (movingSpeed \* secondsPassed);

rectY += (movingSpeed \* secondsPassed);

}

However, there is one problem: if the gap between frame rates are too big (meaning the hardware is too weak) 🡪 the player can have a huge jump.

“To fix this, you'll need to **limit the time factor** to a maximum amount for each frame. By adding the next code to you game loop, your game will never move ahead more than 0.1 second in (game) time. For a game that would normally run at 60fps, this would still mean you've squeezed 6 frames into 1. You can play with this number to make it fit your game.”

// Move forward in time with a maximum amount

secondsPassed = Math.min(secondsPassed, 0.1);

1. [**Resizing canvas dynamically**](https://web.dev/gopherwoord-studios-resizing-html5-games/)