API Chosen: **Gamepad API**

Why did I choose this API ?
The reason is very simple, since we started learning about web programming, an aspect of web programming that we didn't really cover in class was games. We made a scrabble game but we didn't cover how to play it with a controller or how games like pokemon or other 8-bit games could be played using controllers.

My effort in this tutorial is to do exactly that!

Structure of this tutorial:

a)Look into what the gamepad API does and how it works
b)Display how to connect a controller using the gamepad API
c)Make a 8-bit game and use the gamepad API on it

With that being said, let's get started.

Gamepad API:
"HTML5 introduced many of the necessary components for rich, interactive game development"
This includes HTML canvas as well which is excellent for game development. (used for demonstration of the API).
Gamepad is fairly easy to use, and in this tutorial we will explore most of its functionalities.
A list of API commands:
Gamepad API essentially acts as another event listener which tells us if the gamepad is connected on the command:

```
window.addEventListener("gamepadconnected", event =>{});
```

Or disconnected on the command:

```
window.addEventListener("gamepadisconnected", event =>{});
```

However, these 2 commands only help us know when a gamepad is connected or disconnected.
To identify which gamepad is actually being used, we use the command

```
navigator.getGamepads();
```

This displays a list of gamepad objects which is exactly what we want!

What does a gamepad object look like you ask?

```
Gamepad {id: 'Wireless Controller (STANDARD GAMEPAD Vendor: 054c Product: 0ce6)',
index: 0, connected: true, timestamp: 2563, mapping: 'standard', …}
        axes: (4) [0.027451038360595703, 0.027451038360595703, 0.027451038360595703,
        -0.027450978755950928]
```

```
buttons: (18) [GamepadButton, GamepadButton, GamepadButton, GamepadButton,
GamepadButton, GamepadButton, GamepadButton, GamepadButton, GamepadButton,
GamepadButton, GamepadButton, GamepadButton, GamepadButton, GamepadButton,
GamepadButton, GamepadButton, GamepadButton, GamepadButton]
connected: true
id: "Wireless Controller (STANDARD GAMEPAD Vendor: 054c Product: 0ce6)"
index: 0
mapping: "standard"
timestamp: 2563
vibrationActuator: null
[[Prototype]]: Gamepad
```

Now, looking at this, it seems fairly complicated. However, trust me, it would seem easier after I go through the relevant attributes.
Id: fairly straightforward, gives you the id of the controller that you are using.
Axes: gives you the axis in which the controller stick moves. Essentially useful for multi directional movement (like going north-west for example)
Buttons: returns a list of button objects with { pressed, touched,value} as its attributes.

The rest of the keys aren't really that important if you are creating 8-bit 2D games because haptic feedback or vibrations of the controller don't come into play while making it.

Things to look out for:

a) Nintendo switch controllers are not recognized by the api
b) Almost all controllers with the same layout e.g. dualsense 4, dual sense 5 have the same id of *'Wireless Controller (STANDARD GAMEPAD Vendor: 054c Product: 0ce6)'*
c) Haptic feedback from dual sense 5 is not recognized.
d) Connecting the gamepad using bluetooth gave me some issues so I would recommend using a type c wire to connect it.

Tutorial demonstrating the use of gamepad API methods:
Let's put this API into action!
From the repository, let's look at demo.html and demo.js.
The code is fairly simple.

```
console.log("hello world");
window.addEventListener("gamepadconnected",event =>{
   console.log("gamepad is connected:");
   console.log(event.gamepad);
});

window.addEventListener("gamepaddisconnected",event =>{
   console.log("gamepad is disconnected:");
   console.log(event.gamepad);
});
const gamepadDisplay = document.getElementById("gamepad-display");

function update(){
   const gamepads = navigator.getGamepads();
   if(gamepads[0]){
       console.log(gamepads[0]);
       const gamepadState = {
           id: gamepads[0].id,
           buttons:[
                       {button_0:gamepads[0].buttons[0].pressed},
                       {button_1:gamepads[0].buttons[1].pressed},
                       {button_2:gamepads[0].buttons[2].pressed},
                       {button_3:gamepads[0].buttons[3].pressed},
                       {button_4:gamepads[0].buttons[4].pressed},
                       {button_5:gamepads[0].buttons[5].pressed},
                       {button_6:gamepads[0].buttons[6].pressed},
                       {button_7:gamepads[0].buttons[7].pressed},
                       {button_8:gamepads[0].buttons[8].pressed},
                       {button_9:gamepads[0].buttons[9].pressed},
                       {button_10:gamepads[0].buttons[10].pressed},
                       {button_11:gamepads[0].buttons[11].pressed},
                       {button_12:gamepads[0].buttons[12].pressed},
                       {button_13:gamepads[0].buttons[13].pressed},
                       {button_14:gamepads[0].buttons[14].pressed},
                       {button_15:gamepads[0].buttons[15].pressed},
           ]
       };
       gamepadDisplay.textContent = JSON.stringify(gamepadState,null,2);
   }
   window.requestAnimationFrame(update);
}
window.requestAnimationFrame(update);
```

We call the window.addListeners to see if the gamepad is connected or not.
We then call the update function.
This is where the magic happens.

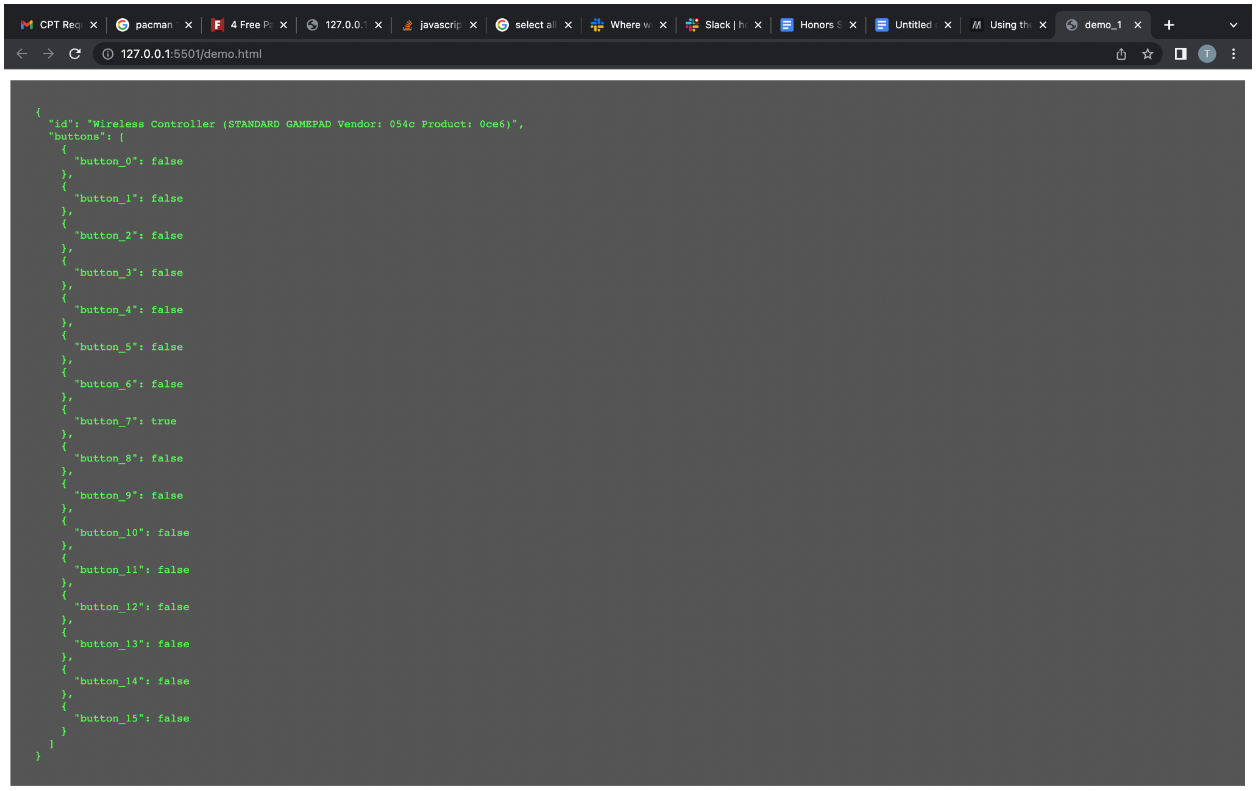
We identify the gamepad we have from `navigator.getGamepads()`

and use its index since navigator.getGamepads() returns an array to identify our controller.

We then filter out the relevant components for us to use by simply returning the id and buttons property.

The important part here to note would be that we are calling window.r**`equestAnimationFrame(update)`** both inside and outside because we are creating an infinite recursive call which I learned is how you render keyframes constantly.

Now, how does this code help us?
I mainly used this to identify which buttons correspond to which id (helpful in our game). This also gives you an idea of how button feedback works and it’s honestly fun to see the buttons change values on being pressed!



```
{
  "id": "Wireless Controller (STANDARD GAMEPAD Vendor: 054c Product: 0ce6)",
  "buttons": [
    {
      "button_0": false
    },
    {
      "button_1": false
    },
    {
      "button_2": false
    },
    {
      "button_3": false
    },
    {
      "button_4": false
    },
    {
      "button_5": false
    },
    {
      "button_6": false
    },
    {
      "button_7": true
    },
    {
      "button_8": false
    },
    {
      "button_9": false
    },
    {
      "button_10": false
    },
    {
      "button_11": false
    },
    {
      "button_12": false
    },
    {
      "button_13": false
    },
    {
      "button_14": false
    },
    {
      "button_15": false
    }
  ]
}
```

Tutorial demonstrating the use of gamepad API on our game
I tried my best to make a 8-bit which was interesting and so I present my version of pac-man. This was somewhat challenging for me personally since
 a) I did not have any idea about designing games.
b) I have never used html canvas.
However, that being said I learned a lot implementing this and so I would like to thank you, the reader, for pushing me to do this project.

I cannot cover html canvas in this tutorial because it’s such a vast topic but I have provided relevant links in the resources section which go over it.

For pac-man the main thing is building the board which is done by a physical array since it comprises an intricate pattern which cannot be randomly assigned.

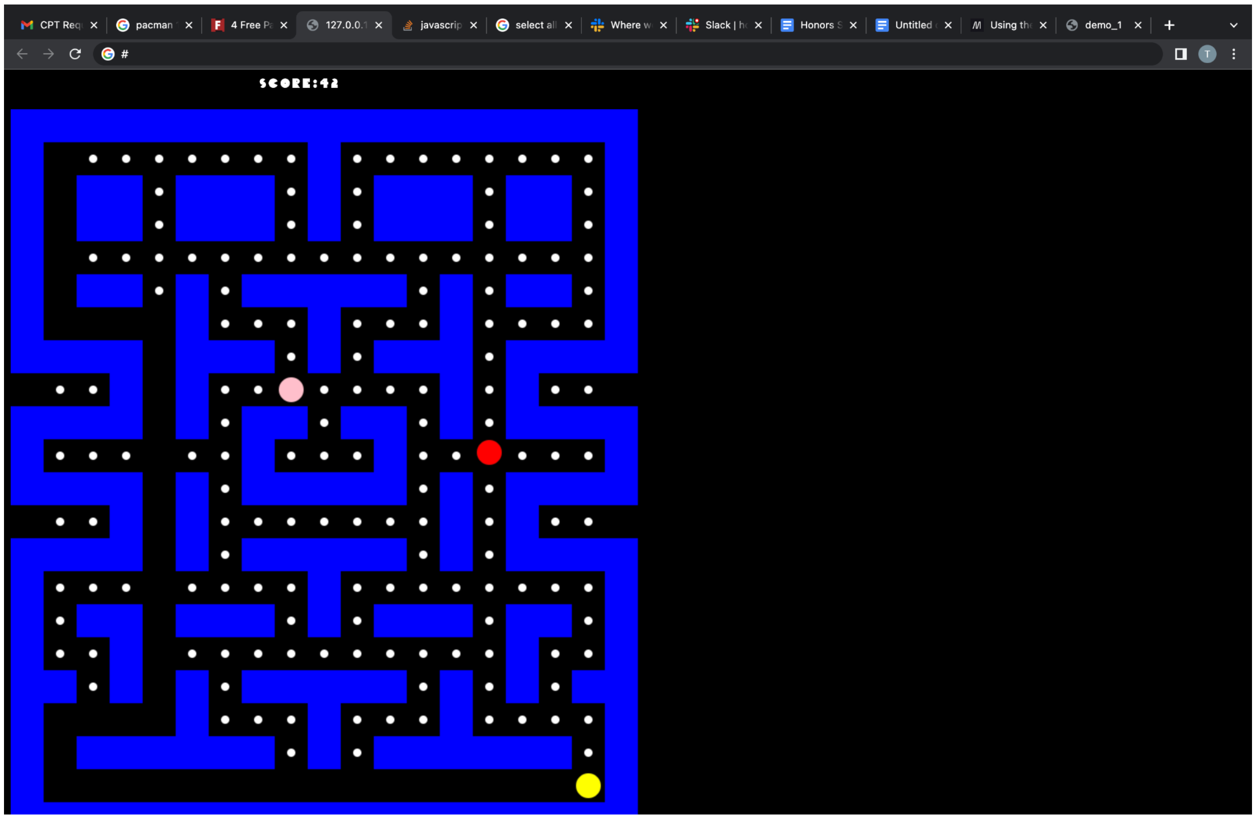
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
   [0, 4, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 4, 0],
   [0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
   [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
   [0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0],
   [0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0],
   [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
   [2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 2, 2, 2],
   [0, 0, 0, 0, 1, 0, 1, 0, 0, 3, 0, 0, 1, 0, 1, 0, 0, 0, 0],
   [0, 2, 2, 2, 1, 1, 1, 0, 3, 3, 3, 0, 1, 1, 1, 2, 2, 2, 0],
   [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
   [2, 2, 2, 0, 1, 0, 1, 1, 1, 2, 1, 1, 1, 0, 1, 0, 2, 2, 2],
   [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
   [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
   [0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
   [0, 4, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 4, 0],
   [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
   [0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0],
   [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
   [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

This is my game board for reference.
We then go over this board using a for each loop and build rectangular blocks (using canvas) where the number = 0.
After that is done, we create the player (pac-man) which has to move along the board.This is where our gamepad api comes into picture. We use our previous project (demo) to figure out which buttons correspond to which button ids.(I used the up,down,left and right on a ps5 controller which mapped to 12,13,14,15). We use these to move the player. How ? you may ask.

We use the infinite animations loop to render the board + player, updating the velocity(position) of the player each time.

This is where the important part comes in. We need to make sure that the player doesn't move past the boundary or stop when it hits the obstacle. This was done by estimating the distance between each obstacle and player using positions and handling the top,left,right and bottom cases respectively.

Similar to the player, we created the ghosts(not intelligent) who randomly pick a direction to go in which does not lead to an obstacle.



Future scope of the API:
This API has become so popular nowadays that people are extending it to have other functionalities. An example would be addition of VR features which track the controller's position.
This is extremely relevant given the direction Meta is taking and I can't be more excited to explore this API in more depth and create more projects (I will make a pokemon:umass region someday hopefully)