1. What is easing? Write few types of easing. Where would you apply easing functions?
   1. Easing helps to specify the speed at which the animation progresses at different points within the animation. Easing makes the animation seem more real as to what’s happening.

Types of Easing:

* Ease-in
* Ease-pit
* Ease-in-out

Ease In –

Ease in animations will start slowly and end fast, this is different to ease-out. This kind of animation is like a heavy stone falling, where it starts slowly and hits the ground quickly with a deadening thump.

Ease-out –

Unlike the ease-in animation, the ease-out animation will start much quicker. But close to the end it will become slower. It is the opposite of ease-in. Ease-out is best used for user interface work. The reason for this is because of the fast start. It will give a feeling of responsiveness.

Ease-in-out –

You make use of both the ease-in and ease-out animations. This can add a more dramatic effect. You can make it so that it starts slowly, in the middle you make it fast then near the end slow it down again. This will increase contract in the animation, which the user can enjoy a lot.

1. What are the pros & cons of using Promises in Javascript? What is the super-set of Promises?
   1. Promises in Javascript are use to handle asynchronous operations in JS. It is much easer to manage when dealing with multiple asynchronous operations, as where callbacks can lead to unmanageable code.

The benefits of using Promises -

* + 1. Promises help improve code readability
    2. As stated before it is better for handling of asynchronous operations
    3. Has better error handling
    4. And better low of control definition in asynchronous logic

The downside of using Promises -

The downside to promises is, that it does not solve what is called the “callback hell”, which is really just a series of nested function calls. Sure, for one call it’s okay. For many calls, your code becomes difficult, it not impossible, to read and maintain.

Super-set of promise –

A lot of promises that is chain.

1. Which non programmable parts of the graphics pipeline can be used to speed up rendering and improve performance. Please give example.

Multi-Sampling –

You can do some render pass at half resolution with multi-sample enabled. In the middle of large triangle geometry the rasterizer will render at a quarter resolution. This of course works if you can reconstruct the missing information by interpolation. So if you rendering depth or normal.

1. Pick three design patterns that are in your opinion the most useful for game development, describe them and write a short example how would you use them in the development process.

Observer –

Very Flexible and expandable way to let classes react to certain things without the need

of tight dependencies due to the face the you can broadcast any event and let anyone who

wants to use the event subscribe to it. The neat thing about the broadcaster is it does not

care about its subscribers. Because of this it will make the system maintainable and expandable and easy to refactor later. So, a basic observes is split into two parts a subject and the observers. The subject will hold a list with all observers interested in getting information(events) when something is happening. When this thing happens, it will send a notification to all the observers. The Observers which are the objects is interested in doing something when an event has happened.

Example –

class Product {

constructor() {

this.price = 0;

this.actions = [];

}

setBasePrice(val) {

this.price = val;

this.notifyAll();

}

register(observer) {

this.actions.push(observer);

}

unregister(observer) {

this.actions.remove.filter(function(el) {

return el !== observer;

});

}

notifyAll() {

return this.actions.forEach(

function(el) {

el.update(this);

}.bind(this)

);

}

}

class fees {

update(product) {

product.price = product.price \* 1.2;

}

}

class profit {

update(product) {

product.price = product.price \* 2;

}

}

export { Product, fees, profit };

Factory –

We can use this pattern to create an object with the purpose of creating other objects from

it. So, if you have a factory class called "GameObject" with static and also possibly parameterized methods to create other game objects like “player", "enemy", "gun", or "bullet". The factor can help take care of the object's complex configuration and thus simply returning a reference to the created object. What this parent does it will help you avoid the complex object instantiation by keeping these complex configurations in a single place.

class CarPattern {

create(type) {

if (type === 'ModelX') return new Car(type, 108000, 300);

if (type === 'ModelS') return new Car(type, 111000, 320);

}

}

class Car {

constructor(model, price, maxSpeed) {

this.model = model;

this.price = price;

this.maxSpeed = maxSpeed;

}

}

export default CarPattern;

Singleton –

This design pattern can only be used and best used to ensure that only a single instance of the object your created can exist at a time. This is normally just used to create a single player game and multiplayer games should be avoided with this pattern design.

Example –

class Person {

constructor() {

if (typeof Person.instance ==='object') {

}

Person.instance = this;

return this;

}

}

export default Person;

1. Explain Redux design pattern.

Redux is really a fairly simple design pattern. All your “write” logic will go into a single function. The only way to run that logic will be, is to give the Redux a plain object that describes something that has happened. The Redux store calls that wirte logic function and passes in the current state tree and the descriptive object, the write logic function returns some new state tree, and the Redux store notifies any subscribers that the state tree has changed.

Redux put some basic constraints on how that write logic function should then work. ATt has to have a signature of (previousState, action) => newState, is known as a reducer function, and must be pure and predictable.

Redux does not really care how you actually structure your logic inside the reducer function, as long as it obeys those basic rules.

1. What is game loop? What is FPS?

A game loop is known as the overall flow control for the entire game program. The game loop will continuously run during the duration of the players gameplay. What it does it is it will process the players input without blocking, updates the game state and renders the game. It tracks the passage of time to control the rate of gameplay. It listens for the player input; it advances the game world and it shows the output on the screen.

Each of theses iterations of the game loop is known as a frame. Most real-time games update several times per second: 30 and 60 as the two most common intervals. So if the game is running on 60FPS, this means that the game loop completes 60 iterations every seconds. So, it is a measurement of performance. Example, FPS is something drawn onto your screen, 60FPS will mean that, that something will be drawn on your screen 60 times I one second. It is a rough measure of how many times you can do your basic game loop per second.

1. What is the difference between time base and frame-based animation?

Time Based Animation –

This is where the movement and the animation depend on the elapsed time. What this mean is that at a given time, the object will be at a specific position displaying a specific frame. Time Based Animation is independent of any frame rate, so that on the slower machines the animation might skip frames but the movements will not slow down. There is a draw back and that is the possible chuncky animation and a much harder collision.

Frame Based Animation –

Frame Based Animation involves using loops to control the elements on the page, allowing more certainty that the code is running when you want to run it. Frame of an animation will not be skipped and the movements is constant. Frame Based Animation depends on a constant FPS. Unlike the Time-Based Animation that will skip frames on a slower machine, the Frame Based Animation will still be able to run it smooth on a closer machine, but it will be slow.

1. Write code to convert an array of strings to an array of the lengths of those strings – use JS ES6 or TS

var stringArr =['A','AA','AAA','AAAA','AAAAA','AAAAAA','AAAAAAA','AAAAAAAA'];

var lengthArr = [];

stringArr.forEach(string => {

length.push(String.length);

});

1. Write code to sum an array of numbers – use JS ES6 or TS

var sum = [1,3,5,7];

sum = sum.reduce(function(acc, value) {

return acc + value;

}, 0)

console.log(sum);