

# Phaser 3

## Tecnologias Multimédia

# Phaser

## What is Phaser?



Phaser is an open-source framework for building 2D games with JavaScript and HTML5.

It was created by Richard Davey in 2013 and has gone through several major updates and improvements over the years. Phaser has been widely used to create games of all genres and has played an important role in making game development accessible to a wider audience.

The latest version, Phaser 3, was released in 2018 and represents a major overhaul of the framework with new features and improved performance. Overall, Phaser's history is one of innovation, collaboration, and community.



# Phaser 3

## What is Phaser 3?



Phaser 3 is a popular open-source game development framework for creating 2D games for web browsers and mobile devices using HTML5, JavaScript, and WebGL.

It provides a rich set of features and tools for creating games, including asset loading, rendering, physics, input handling, audio, and more.

Phaser 3 is known for its ease of use, flexibility, and robustness, and has a large and active community of developers and contributors.





WebGL stands for Web Graphics Library, and it is a JavaScript API for rendering interactive 3D and 2D graphics within web browsers.

WebGL is based on OpenGL ES, a widely used 3D graphics API for mobile devices, and allows developers to use low-level graphics programming to create complex and visually stunning graphics directly in the browser, without the need for plugins or external software. WebGL is supported by most modern web browsers, including Chrome, Firefox, Safari, and Edge, and is commonly used in game development, virtual reality, and scientific visualization.



# Canvas

## What is Canvas?




Canvas is a built-in HTML5 element that provides a 2D drawing surface for JavaScript code.

It allows developers to dynamically create and manipulate graphics on a web page using JavaScript code, without the need for additional plugins or software.

With Canvas, developers can draw lines, circles, rectangles, text, and images, and manipulate them in real-time to create interactive web content, including games, animations, data visualizations, and more.





The main difference between WebGL and Canvas is that WebGL is a rendering technology based on OpenGL, a cross-platform graphics API, while Canvas is a 2D drawing API built into HTML5. Here are some more specific differences:

1. 3D vs. 2D: WebGL is designed specifically for rendering 3D graphics, while Canvas is primarily used for 2D graphics.
2. Performance: WebGL is generally faster and more efficient than Canvas for complex and interactive graphics, due to its use of hardware acceleration and advanced rendering techniques.
3. Complexity: WebGL requires a higher level of technical expertise and knowledge of 3D graphics programming than Canvas, which can be used by developers with a wide range of skill levels.
4. Support: WebGL is not supported by all web browsers and devices, while Canvas has widespread support across all major browsers.
5. Features: WebGL supports advanced features such as shader programming and advanced lighting effects, while Canvas has a more limited set of features for drawing 2D shapes and images.

# WebGL and Canvas

## In summary...



Instituto Politécnico  
de Viana do Castelo



WebGL and Canvas serve different purposes and have different strengths and weaknesses.

WebGL is ideal for rendering complex 3D graphics, while Canvas is best suited for 2D graphics and simple animations

## Phaser 3

### Canvas and WebGL



Instituto Politécnico  
de Viana do Castelo



Phaser 3 can use both the HTML5 Canvas and WebGL rendering technologies.

By default, Phaser 3 uses WebGL to render games, as it provides better performance and more advanced features such as shader effects and 3D rendering. However, if a user's device or browser does not support WebGL, Phaser 3 can automatically fall back to using the Canvas renderer instead.

This makes Phaser 3 a versatile framework that can run on a wide range of devices and platforms.





## Phaser 3

### Why?



Instituto Politécnico  
de Viana do Castelo



Phaser 3 is a popular choice for developing 2D games for the web for several reasons:

1. **Ease of Use:** Phaser 3 has a user-friendly API that makes it easy to create 2D games, even for developers who are new to game development.
2. **Robustness:** Phaser 3 provides a wide range of features and tools for creating games, including asset loading, rendering, physics, input handling, audio, and more.
3. **Cross-platform Support:** Phaser 3 games can be developed once and deployed on multiple platforms, including desktop and mobile devices.
4. **Performance:** Phaser 3 is optimized for performance and can render large numbers of sprites and animations at high frame rates, even on low-powered devices.
5. **Community Support:** Phaser 3 has a large and active community of developers and contributors who provide support, resources, and tutorials to help new developers get started and improve their skills.

Overall, Phaser 3 provides an efficient and effective way to create 2D games for the web, and its popularity is a testament to its ease of use, performance, and robustness.

# Phaser 3 Requirements



Instituto Politécnico  
de Viana do Castelo



To start working with Phaser 3, you need to have the following software requirements:

1. Node.js: Phaser 3 requires Node.js to be installed on your computer to be able to use npm to install Phaser and other dependencies.
2. A code editor: You can use any code editor of your choice, but popular option include Visual Studio Code.
3. A web browser: You will need a web browser to test your Phaser 3 game. Phaser 3 supports all modern browsers including Chrome, Firefox, Safari, and Edge.
4. A local web server: Phaser 3 requires a local web server to run your game. You can use a simple web server like http-server or live-server, or a more robust server like Apache or Nginx.



In Phaser 3, a sprite is a game object that represents a 2D image or animation that can be moved, rotated, and scaled on the screen. Sprites are a fundamental element of most 2D games and can represent characters, objects, or other elements in the game world.

In Phaser 3, a sprite is created using the ***this.add.sprite()*** method and can be customized with different properties, such as the sprite's position, scale, rotation, and animation frames. Sprites can also be given physical properties, such as mass, velocity, and acceleration, which allow them to interact with other game objects and the game world.

Phaser 3 provides a rich set of APIs for working with sprites, including support for animation, physics, and user input. With these features, developers can create engaging and dynamic 2D games with ease.

## Phaser 3

### Concepts – Images – Load and display



1- First, load the images you want to use in your game using Phaser's ***this.load.image()*** method:

```
this.load.image('player', 'assets/images/player.png');
```

2 - Create a sprite: Next, create a sprite to display the image on the screen. You can create a sprite using Phaser's ***this.add.sprite()*** method:

```
let player = this.add.sprite(0, 0, 'player');
```

3 - Customize the sprite: You can customize the sprite with different properties, such as position, scale, and rotation:

```
player.setPosition(this.cameras.main.centerX, this.cameras.main.centerY);
```

4 - Add the sprite to the game: Finally, add the sprite to the game using the ***this.add.existing()*** method:

```
this.add.existing(player);
```

## Phaser 3

### Concepts – Sprite animations



1 - Load the sprite sheet: First, load the sprite sheet that contains the frames of the animation using the `this.load.spritesheet()` method.

```
this.load.spritesheet('player', 'assets/images/player.png', {  
    frameWidth: 32,  
    frameHeight: 32  
});
```

2 - Create the sprite: Next, create a sprite using the loaded sprite sheet. You can use the `this.add.sprite()` method and pass the x and y position of the sprite, as well as the name of the sprite sheet.

```
let player = this.add.sprite(100, 100, 'player');
```

## Phaser 3

### Concepts – Sprite animations cont.



Define the animation: Define the animation by using the ***this.anims.create()*** method and passing an object that describes the animation. The object should have a unique key, a reference to the sprite sheet, and an array of frame numbers or strings that define the sequence of frames to play.

```
this.anims.create({  
  key: 'player_walk',  
  frames: this.anims.generateFrameNumbers('player', { start: 0, end: 3 } ),  
  frameRate: 10,  
  repeat: -1  
});
```

Play the animation: Finally, play the animation on the sprite using the ***play()*** method.

```
player.play('player_walk');
```

## Phaser 3

### Concepts – Collisions



To detect collisions and interactions between objects in Phaser 3, you can follow these steps:

1 - Enable physics: First, enable physics in your game using the ***this.physics.add*** method.

```
this.physics.add.arcade();
```

2 - Create objects: Create the objects that you want to check for collisions and interactions.

```
let player = this.physics.add.sprite(100, 100, 'player');
```

```
let enemy = this.physics.add.sprite(200, 200, 'enemy');
```

3 - Define collisions: Define the collisions by using the ***this.physics.add.collider*** method and passing the objects that you want to collide.

```
this.physics.add.collider(player, enemy);
```



4 - Define interactions: Define the interactions by using the ***this.physics.add.overlap*** method and passing the objects that you want to interact.

```
this.physics.add.overlap(player, collectible, collectItem, null, this);
```

5 - Handle collisions and interactions: Finally, handle the collisions and interactions by creating functions that are called when the events occur.

```
function onCollide(player, enemy) {  
    // Do something when the player and enemy collide  
}
```

6 - Similarly, to handle the interaction between the player and the collectible item.

```
function collectItem(player, collectible) {  
    collectible.disableBody(true, true);  
    // Do something when the player collects the item  
}
```



## Phaser 3

### Before start...



```
var config = {
  type: Phaser.AUTO,
  width: 800,
  height: 600,
  scene: {
    preload: preload,
    create: create,
    update: update
  }
};

var game = new Phaser.Game(config);

function preload ()
{
}

function create ()
{
}

function update ()
{
}
```

## Phaser 3

### Preload, create and update



**preload():** This function is used to load the game assets, such as images, audio, and fonts, before the game starts. This function is called only once during the game's lifecycle, usually at the beginning, to load all the assets that are needed for the game.

**create():** This function is used to create the game objects and initialize the game state after the assets have been loaded. This function is also called only once during the game's lifecycle, usually after the preload() function has finished. In this function, you can create the game sprites, add them to the scene, and set their initial properties.

**update():** This function is called every frame of the game and is used to update the game state based on the user input or any other game logic. In this function, you can update the positions of the game objects, check for collisions and interactions, and handle any other game events.

Overall, these three functions work together to create a complete game loop in Phaser 3, where the preload() function loads the assets, the create() function sets up the game, and the update() function handles the game logic and state updates on each frame.

---

<https://phaser.io/learn>





<https://phaser.io/tutorials/getting-started-phaser3>

Good  
Luck!

---

A hand-drawn smiley face consisting of a circle with two dots for eyes and a curved line for a mouth.



<https://phaser.io/tutorials/making-your-first-phaser-3-game>

Good  
Luck!

---

o teu • de partida



Instituto Politécnico  
de Viana do Castelo

[www.ipvc.pt](http://www.ipvc.pt)