Tekst til «The Game»

Dette er et spill jeg skal skrive mest mulig selv.

Innholdsfortegnelse

[A high-level step-by-step guide for creating a platformer game: 2](#_Toc135822809)

[Kort om JavaScript: 3](#_Toc135822810)

[1. Understand the basics: 3](#_Toc135822811)

[2. Use conditional statements: 3](#_Toc135822812)

[3. Use loops: 3](#_Toc135822813)

[4. Use functions: 3](#_Toc135822814)

[5. Understand return values: 4](#_Toc135822815)

[6. Use events: 4](#_Toc135822816)

[7. Practice: 4](#_Toc135822817)

[8. Test your knowledge: 4](#_Toc135822818)

[Kort fortalt: 5](#_Toc135822819)

[Eksempel: 6](#_Toc135822820)

[// Define a function that calculates the sum of two numbers 6](#_Toc135822821)

[// Call the sum function and store the result in a variable 6](#_Toc135822822)

[// Use a conditional statement to check if the result is greater than 5 6](#_Toc135822823)

[// Use a for loop to iterate over an array of numbers 6](#_Toc135822824)

[// Use an event listener to listen for clicks on a button 6](#_Toc135822825)

[Conditional statements: 7](#_Toc135822826)

[Loops: 8](#_Toc135822827)

[Functions: 9](#_Toc135822828)

[Return value: 10](#_Toc135822829)

[Events: 11](#_Toc135822830)

# A high-level step-by-step guide for creating a platformer game:

1. **Plan your game**: Start by planning out your game. What is the story or theme? What are the gameplay mechanics? What are the levels like? What are the characters and enemies like? Create a design document that outlines all of these details.
2. **Create the art**: Create the art for your game, including the characters, enemies, backgrounds, and user interface elements. You can create the art yourself or work with an artist.
3. **Choose a game engine**: Choose a game engine to develop your game in. There are many game engines available, including Unity, Unreal Engine, and Godot. Choose one that fits your needs and skill level.
4. **Implement the gameplay mechanics**: Use the game engine to implement the gameplay mechanics of your game. This will involve writing code to control the player character, enemies, and other elements of the game.
5. **Create the levels**: Use the game engine’s level editor to create the levels for your game. Place the backgrounds, platforms, enemies, and other elements to create challenging and fun levels.
6. **Test and iterate**: Playtest your game regularly to identify and fix any issues. Iterate on the gameplay mechanics and levels to make them as fun and engaging as possible.
7. **Polish and release**: Once your game is complete, spend some time polishing it to make it as good as it can be. Then release it on your chosen platform(s) for others to play.

# A step-by-step plan for structuring your JavaScript code when creating a platformer game:

1. **Organize your code**: Start by organizing your code into separate files and folders based on their functionality. For example, you might have a folder for game logic, a folder for user interface code, and a folder for utility functions.

* When creating a game or any other complex project, it’s important to keep your code organized and easy to navigate. One way to do this is by organizing your code into separate files and folders based on their functionality. For example, you might have a folder for game logic that contains all the code related to the core gameplay mechanics.
* You might also have a folder for user interface code that contains all the code related to menus, HUDs, and other UI elements. Additionally, you might have a folder for utility functions that contains reusable code snippets that can be used throughout your project. Organizing your code in this way can make it easier to find and edit specific parts of your codebase.

1. **Define your game objects**: Define the objects that make up your game, such as the player character, enemies, platforms, and power-ups. Create classes or constructor functions for each object type and define their properties and methods.

* In this step, you should define the objects that make up your game, such as the player character, enemies, platforms, and power-ups. You can create classes or constructor functions for each object type and define their properties and methods.
* For example, you might have a Player class that defines properties such as the player’s position, health, and inventory. You could also define methods for the Player class that handle actions such as moving, jumping, and attacking.
* Similarly, you could create classes for other game objects such as enemies and platforms. Each class should define the properties and behaviors that are specific to that object type.
* By defining your game objects in this way, you can create a modular codebase where each object type is self-contained and can be easily reused throughout your game.

1. **Implement the game loop**: Implement the main game loop that runs every frame and updates the game state. The game loop should handle user input, update the positions of game objects, check for collisions, and render the game to the screen.

* The game loop is the core of any game and is responsible for updating the game state and rendering the game to the screen. In this step, you should implement the main game loop that runs every frame.
* The game loop typically consists of several steps that are executed in order. These steps might include:

1. **Handle user input**: Process any user input that was received since the last frame. This could include moving the player character or firing a weapon.
2. **Update game objects**: Update the positions and states of all game objects. This could include moving enemies, animating sprites, or applying physics.
3. **Check for collisions**: Check for collisions between game objects and resolve them. This could include detecting when the player character collides with an enemy or a platform.
4. **Render the game**: Draw the game to the screen. This could include drawing the game world, characters, and UI elements.

* By implementing a game loop that performs these steps every frame, you can create a smooth and responsive gameplay experience.

1. **Handle user input**: Write code to handle user input from the keyboard, mouse, or game controller. Use event listeners to listen for specific input events and update the game state accordingly.
2. **Implement game mechanics**: Implement the core gameplay mechanics of your game, such as jumping, shooting, and collecting power-ups. Write functions to handle these mechanics and call them from the appropriate places in your code.
3. **Create levels**: Create data structures to represent the levels in your game. Each level should include information about the layout of platforms, enemies, and power-ups. Write code to load levels from these data structures and create the corresponding game objects.
4. **Add polish**: Add polish to your game by implementing features such as animations, sound effects, and particle effects. Write code to handle these features and integrate them into your game.

# Kort om JavaScript:

Hentet fra en nettside:

[Making decisions in your code — conditionals - Learn web development | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/conditionals)

Based on the current web page context, here is a simple guide to creating your own game using JavaScript:

1. Understand the basics: Make sure you have some familiarity with HTML and CSS and have worked through the previous module, JavaScript first steps.

* This will provide you with a solid foundation to build upon as you learn more advanced concepts in JavaScript. If you need to brush up on your HTML and CSS skills or work through the JavaScript first steps module, you can find resources to help you do so.

1. Use conditional statements: In any programming language, code needs to make decisions and carry out actions accordingly depending on different inputs. For example, in a game, if the player’s number of lives is 0, then it’s game over.

* Conditional statements are used to perform different actions based on different conditions. For example, in a game, if the player’s number of lives is 0, then it’s game over. In a weather app, if it is being looked at in the morning, show a sunrise graphic; show stars and a moon if it is night.
* Conditional statements allow you to control the flow of your program and make it more dynamic.

1. Use loops: Sometimes you need a task done more than once in a row. For example, looking through a list of names. In programming, loops perform this job very well.

* Loops are used to repeat a specific block of code a known or unknown number of times, depending on the situation. There are several types of loops in JavaScript, including for, while, and do...while loops.
* Each type of loop has its own syntax and use cases. Loops can help you automate repetitive tasks and make your code more efficient.

1. Use functions: Functions allow you to store a piece of code that does a single task inside a defined block and then call that code whenever you need it using a single short command. This can help you organize your code and make it more readable.

* Functions are another essential concept in coding. Functions allow you to store a piece of code that does a single task inside a defined block and then call that code whenever you need it using a single short command — rather than having to type out the same code multiple times.
* This can help you organize your code, make it more readable, and reduce the likelihood of errors. In JavaScript, functions can be defined using function declarations or function expressions, and they can accept parameters and return values. Functions can also have scope, which determines where in your code the function can be accessed and used.

1. Understand return values: Some functions return a significant value after completion, while others do not. It’s important to understand what their values are and how to make use of them in your code.

* When a function returns a value, that value can be used in other parts of your code. For example, you might have a function that calculates the sum of two numbers and returns the result. You could then use that returned value in a different part of your code to perform another calculation or display the result to the user. Understanding return values can help you write more flexible and reusable code.

1. Use events: Events are actions or occurrences that happen in the system you are programming. For example, if the user clicks a button on a webpage, you might want to respond to that action by displaying an information box.

* Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired. For example, if the user clicks a button on a webpage, you might want to respond to that action by displaying an information box.
* In JavaScript, you can use event listeners to listen for specific events and execute code when those events occur. There are many different types of events, including user-initiated events like clicks and key presses, and system-initiated events like page load and resize. Understanding and using events can help you create more interactive and dynamic web pages and applications.

1. Practice: The page provides practical experience with building up your own custom function and understanding return values.

* The page provides practical experience with building up your own custom function. Along the way, it also explains some further useful details of dealing with functions.
* Practicing building your own custom functions can help you better understand how they work and how to use them effectively in your code. It’s important to not just read about concepts, but to also apply them in practice to reinforce your understanding and build your skills.

1. Test your knowledge: The page includes an assessment to test your understanding of the JavaScript basics covered in the guides.

* The page includes an assessment to test your understanding of the JavaScript basics covered in the guides. The assessment involves building a fairly common item you’ll see on a lot of websites — a JavaScript-powered image gallery.
* Taking the assessment can help you gauge your understanding of the concepts covered in the module and identify areas where you may need further practice or study. Testing your knowledge is an important part of the learning process and can help you track your progress and improve your skills.

I hope this simple guide helps you get started with creating your own game using JavaScript! 😊

## Kort fortalt:

1. **Understand the basics**: Make sure you have some familiarity with HTML and CSS and have worked through the previous module, JavaScript first steps.
2. **Use conditional statements**: Control the flow of your program by making decisions and carrying out actions based on different conditions.
3. **Use loops**: Automate repetitive tasks by repeating a specific block of code a known or unknown number of times.
4. **Use functions**: Organize your code and make it more readable by storing pieces of code that do single tasks inside defined blocks and calling them whenever needed.
5. **Understand return values**: Know what values functions return and how to make use of them in your code.
6. **Use events**: Create more interactive and dynamic web pages and applications by responding to actions or occurrences that happen in the system you are programming.
7. **Practice**: Build up your own custom function and understand return values through practical experience.
8. **Test your knowledge**: Gauge your understanding of the concepts covered in the module by taking an assessment that involves building a JavaScript-powered image gallery.

# Eksempel:

Here is an example of a simple JavaScript code that demonstrates some of the concepts we discussed:

## // Define a function that calculates the sum of two numbers

function sum(a, b) {

return a + b;

}

## // Call the sum function and store the result in a variable

let result = sum(3, 4);

console.log(result); // 7

## // Use a conditional statement to check if the result is greater than 5

if (result > 5) {

console.log('The result is greater than 5');

} else {

console.log('The result is not greater than 5');

}

## // Use a for loop to iterate over an array of numbers

let numbers = [1, 2, 3, 4, 5];

for (let i = 0; i < numbers.length; i++) {

console.log(numbers[i]);

}

## // Use an event listener to listen for clicks on a button

let button = document.querySelector('button');

button.addEventListener('click', function() {

console.log('The button was clicked');

});

This code defines a sum function that calculates the sum of two numbers and returns the result. The function is then called with the arguments 3 and 4, and the result is stored in a variable. A conditional statement is used to check if the result is greater than 5 and display a message accordingly. A for loop is used to iterate over an array of numbers and log each number to the console. Finally, an event listener is added to a button element that listens for clicks and logs a message to the console when the button is clicked.

# Conditional statements:

A simple JavaScript code that demonstrates the use of a conditional statement:

// Define a variable to store the player's score

let playerScore = 85;

// Use a conditional statement to check the player's score

if (playerScore >= 90) {

console.log('Congratulations! You got an A!');

} else if (playerScore >= 80) {

console.log('Good job! You got a B!');

} else if (playerScore >= 70) {

console.log('Not bad! You got a C!');

} else if (playerScore >= 60) {

console.log('You passed with a D!');

} else {

console.log('Sorry, you failed.');

}

In this example, we define a variable playerScore to store the player’s score. We then use a conditional statement to check the value of playerScore and display a message accordingly. If the player’s score is greater than or equal to 90, we display a message congratulating them for getting an A. If the player’s score is greater than or equal to 80 but less than 90, we display a message telling them they got a B. We continue checking for different ranges of scores and displaying appropriate messages. If the player’s score is less than 60, we display a message telling them they failed.

I hope this example helps you understand how conditional statements can be used in practice! 😊

# Loops:

A simple JavaScript code that demonstrates the use of a loop:

// Define an array of numbers

let numbers = [1, 2, 3, 4, 5];

// Use a for loop to iterate over the array of numbers

for (let i = 0; i < numbers.length; i++) {

console.log(numbers[i]);

}

// Use a while loop to iterate over the array of numbers

let i = 0;

while (i < numbers.length) {

console.log(numbers[i]);

i++;

}

In this example, we define an array of numbers called numbers. We then use two different types of loops to iterate over the array and log each number to the console. The first loop is a for loop, which uses a counter variable i to keep track of the current iteration. The loop starts with i equal to 0 and continues until i is no longer less than the length of the numbers array. On each iteration of the loop, we log the current number to the console and increment i by 1.

The second loop is a while loop, which continues to execute as long as its condition is true. In this case, the condition is that i is less than the length of the numbers array. On each iteration of the loop, we log the current number to the console and increment i by 1. The loop continues until i is no longer less than the length of the numbers array.

# Functions:

A simple JavaScript code that demonstrates the use of a function:

// Define a function that calculates the area of a rectangle

function calculateArea(width, height) {

let area = width \* height;

return area;

}

// Call the calculateArea function and store the result in a variable

let result = calculateArea(5, 10);

console.log(result); // 50

In this example, we define a function called calculateArea that calculates the area of a rectangle. The function takes two parameters: width and height. Inside the function, we calculate the area by multiplying width by height and store the result in a variable called area. We then use the return keyword to return the value of area from the function.

After defining the function, we call it by passing in two arguments: 5 and 10. These arguments correspond to the width and height parameters of the calculateArea function. The function calculates the area of a rectangle with a width of 5 and a height of 10, which is 50, and returns that value. We store the returned value in a variable called result and log it to the console.

# Return value:

A simple JavaScript code that demonstrates the use of a return value:

// Define a function that calculates the sum of two numbers

function sum(a, b) {

let result = a + b;

return result;

}

// Call the sum function and store the result in a variable

let result = sum(3, 4);

console.log(result); // 7

// Use the returned value in another calculation

let doubleResult = result \* 2;

console.log(doubleResult); // 14

In this example, we define a function called sum that calculates the sum of two numbers. The function takes two parameters: a and b. Inside the function, we calculate the sum by adding a and b together and store the result in a variable called result. We then use the return keyword to return the value of result from the function.

After defining the function, we call it by passing in two arguments: 3 and 4. These arguments correspond to the a and b parameters of the sum function. The function calculates the sum of 3 and 4, which is 7, and returns that value. We store the returned value in a variable called result and log it to the console.

We can then use the returned value in another calculation. In this case, we multiply the returned value by 2 to get 14, and store that value in a variable called doubleResult. We then log doubleResult to the console.

Events:  
A simple JavaScript code that demonstrates the use of an event:

// Get the button element from the page

let button = document.querySelector('button');

// Add an event listener to the button that listens for clicks

button.addEventListener('click', function() {

// This code will be executed when the button is clicked

console.log('The button was clicked');

});

In this example, we use the document.querySelector method to get a reference to a button element on the page. We then use the addEventListener method to add an event listener to the button that listens for click events. The addEventListener method takes two arguments: the type of event to listen for ('click' in this case) and a function to execute when that event occurs.

The function we pass as the second argument to addEventListener is called an event handler. This function will be executed whenever the specified event occurs on the target element. In this case, our event handler logs a message to the console whenever the button is clicked.

# Key takeaways (from webpage)

**Conditional statements**

* Conditional statements allow us to run different code depending on the result of a test condition.
* The most common type of conditional statement is if…else, which has a condition, a code block that runs if the condition is true, and another code block that runs if it is false.
* We can also use else if to add more choices, and switch to handle many choices in a more readable way.

**Comparison operators**

* Comparison operators are used to test the conditions inside conditional statements.
* They include === (equal to), !== (not equal to), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to).

**Logical operators**

* Logical operators allow us to combine multiple conditions or negate them.
* They include && (AND), || (OR), and ! (NOT).

**Ternary operator**

* The ternary operator is a shorthand way of writing a simple if…else statement.
* It takes a condition and two expressions, and returns the first expression if the condition is true, and the second expression if the condition is false.