Game Of Dragons

Semesteroppgave 2 by Bjørn-Ivar Skuggen

In this assignment we got the task to create a board-game using HTML/CSS and JavaScript. The task was to create a logo, tokens for the players, design a landing page, board-game page, and a winner page.

Logo (seen at top of document)

The game is named “Game Of Dragons”, and the logo has to represent exactly that. Therefore, the design is minimalistic and straight to the point – two dragon wings surrounding the name of the game. The cartoon-look of the wings are made to match the tokens in the game, and the red logo represents the danger of dragons.

Tokens / Characters

The characters are chosen from the provided API. To create the tokens for these characters, I started off by finding an image that would fit my game well. Further,I gave the whole image a cartoon-like effect, cropped out the face of the character, created a black/white layer, masked out the background, and finally added the gradient background. Some tokens also got a little makeover by the brush tool. This way I got to use several of the techniques from Design 2. I wanted the characters to appeal to both children and adults, which is the reason I created them as cartoons. Still, I wanted them to be recognizable from the series, so I kept their traits

very visible. I am aware of copyright for the tokens could be an issue, but I have edited them away from their original state, and removed background, colors, and the realistic look of their faces. As the information on this matter is a bit diffuse, I decided to go with my original idea.

Landing page

The landing page is pretty simple. To give the site a little suspense from the start, the only thing that shows for the first four seconds is a quote by Daenerys Targaryen on a dark video background of sparks and flames. After four seconds, the “Start Game” button appears, and the game is ready to begin.

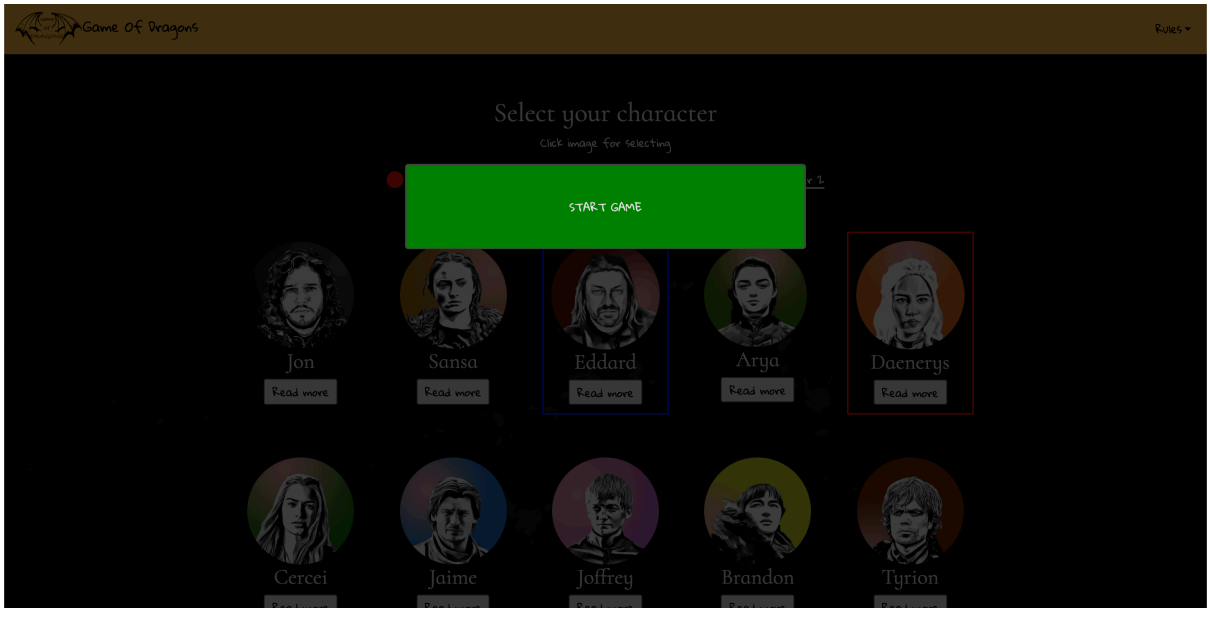
SelectCharacter page

In the SelectCharacter page, the users select their character for the game. When clicking “Read more”, the user can read the information provided from the API in a pop-up modal. To select the character, the user simply clicks the image of their desired character, and are presented with the option to confirm or deny their choice. If user clicks “Nope” they get to choose another character, if they click “YEAH” the character is sent to sessionStorage and it is players 2s turn to choose.

The layout is simple: Image —> Name —> Read more.

The user should never be in doubt of which character they are looking at.

When both players have chosen their character, the information is passed to the sessionStorage, and the “Start Game” button is displayed.

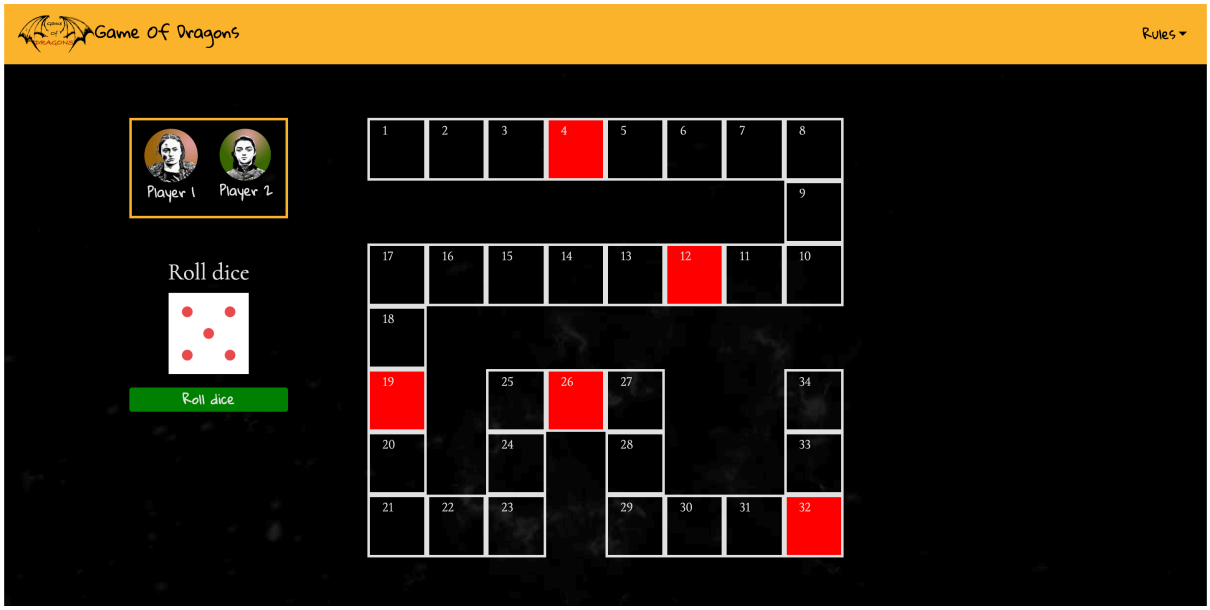


Game page

The game page is where the fun starts. The players roll the dice, the characters move, they get trapped, and there will be a winner. The layout of the site is made to make the user understand the path of the trail and to understand where the goal is. The traps are easily spotted, and a red modal pops up when someone hits them. The tokens move around the board, and when one of them reaches the goal, the winner modal pops up. I chose a modal instead of a new site because it fits better in my whole site as I am using modals for most of the happenings throughout the site.

Both the board itself and the dice have sharp edges and are square, which is meant as a contrast too many of the other components in the site. The visual look is also more pleasing when the tiles are put together without any distance between the edges.

All the sites are created so you can tab your way through the site. Also, WCAG tests has been done.



Fonts:

Fonts that are being used are Cinzel Decorative for the main text, and Gloria Hallelujah for buttons, rules, and small sentences and words. The Cinzel text is to get the old Game Of Thrones-feeling, while the Gloria font is added to give the game-feeling and keep it still childish.

The code

When I first started this project, I got some fast results because of shortcuts and going for solutions that “just works”. After about two weeks in, the easy solutions and quick fixes finally caught up to me. Unstructured and unreadable code made the whole project fall apart as I could not find a way to make it work. I therefore had to start all over again.

On the second try, everything went a little better. Now, I had a deeper understanding of how to structure the code, and to make it understandable. I kept some of the ideas from the first code but made sure it would make sense to use it.

```
function game() {
    // Create Dice and spin it
    document.querySelector('.btn-roll').addEventListener('click', function () {
        var diceDOM = document.querySelector('.dice');
        // Spin the dice
        if (diceDOM.classList.contains('rotate')) {
            diceDOM.classList.remove('rotate');
        } else {
            diceDOM.classList.add('rotate');
        }

        if (gamePlaying) {
            // 1. Random number
            var dice = Math.floor(Math.random() * 6) + 1;

            //2. Display the result in DOM
            setTimeout(function () {
                diceDOM.src = 'graphics/img/dice/dice-' + dice + '.png';
            }, 250);


            if (dice) {
                //Add score
                diceRollScore = dice;
                scores[activePlayer] += diceRollScore;
                console.log("player 1: " + scores[0]);
                console.log("player 2: " + scores[1]);

                // If dice rolled a 6, dont change the activePlayer as the same player gets another roll
                function ifDiceRolledSix() {
                    if (diceRollScore == 6) {
                        activePlayer;
                    }
                    // Change the player if anything else than 6 is rolled
```

The old code contained functions inside functions. It all turned out to be a mess.

Selecting the characters

To select the characters, I decided to use a switch statement. This is because I can create a case for each image clicked and pass in the characters URL ending to the function setCorrectPlayerUrl. I will then be able to pass the information further into sessionStorage at a later point.

```
function choosePlayer(clicked) {
    switch (clicked) {

        case "selectJon": // if the player clicks on "selectJon" image, give setCorrectPlayerUrl the paramater of 583, which is Jon´s url ending
            setCorrectPlayerUrl("583"); // This is where the characterId is passed in
            break;
        case "selectSansa":
            setCorrectPlayerUrl("957"); // This is where the characterId is passed in
            break;
        case "selectEddard":
            setCorrectPlayerUrl("339"); // This is where the characterId is passed in
            break;
        case "selectArya":
            setCorrectPlayerUrl("148"); // This is where the characterId is passed in
            break;
        case "selectDaenerys":
            setCorrectPlayerUrl("271"); // This is where the characterId is passed in
            break;
        case "selectCersei":
            setCorrectPlayerUrl("238"); // This is where the characterId is passed in
            break;
        case "selectJaime":
            setCorrectPlayerUrl("529"); // This is where the characterId is passed in
            break;
        case "selectJoffrey":
            setCorrectPlayerUrl("565"); // This is where the characterId is passed in
            break;
        case "selectBrandon":
            setCorrectPlayerUrl("206"); // This is where the characterId is passed in
            break;
        case "selectTyrion":
            setCorrectPlayerUrl("1052"); // This is where the characterId is passed in
            break;
    }
}
```

The characterId from choosePlayer is passed into setCorrectPlayerUrl, and here I display the modal with the confirm or deny options. I use a modal for this so the user can think twice about who they choose. If they press “YEAH” there is no way back.

```
// Set the correct URL to the current players choice of character
function setCorrectPlayerUrl(characterId) {
    if (player1Active) {
        player1Character = apiUrl + characterId; // Creates an url with correct id
        console.log("player1 " + player1Character);
        $('#confirmModal').modal('show'); // Show confirm modal
        document.getElementById("confirm").id = ("confirm-" + characterId); // Change id of "confirm"
        document.getElementById("deny").id = ("deny-" + characterId); // Change id of "deny"
    } else {
        player2Character = apiUrl + characterId;
        console.log("player2 " + player2Character);
        $('#confirmModal').modal('show');
        document.getElementById("confirm").id = ("confirm-" + characterId);
        document.getElementById("deny").id = ("deny-" + characterId);
    }
}
```

When the modal is displayed, the user has to make a final choice; to accept the current player, or to change to another one. The click of either of the buttons trigger the confirmDeny function. If the user clicks confirm (YEAH), the characterId is sent to sessionStorage, and some styling is added to the chosen character. It also changes player if confirm is clicked. If deny (Nope) is clicked, it removes modal and gives the player the opportunity to choose again.

```
// Optiond for the confirm/deny buttons
function confirmDeny(clicked){
    let splitClicked = clicked.split("-"); // Remove the "-" from id name and creates array ["confirm", "id"]
    let decision = splitClicked[0];
    let characterId = splitClicked[1];
    let cardCharacterId = document.getElementById("card" + characterId);
    let changeConfirmId = document.getElementById("confirm-" + characterId);
    let changeDenyId = document.getElementById("deny-" + characterId);

    if(decision == "confirm"){ // Check if confirmed is clicked
        if(player1Active){
            sessionStorage.setItem("player1", characterId); // Send player1 id to sessionstorage
            cardCharacterId.style.padding = "12px"; // Add padding to selected player1 character
            cardCharacterId.style.border = "2px solid red"; // Add border to selected player1 character
            changeConfirmId.id = ("confirm"); // Change confirm button id back to "confirm"
            changeDenyId.id = ("deny"); // Change deny button id back to "deny"
            $('#confirmModal').modal('hide'); // Hide modal
            player1Active = false; // Set player1 to false, so player2 is active
            activeClass(); // Change active player visually
        } else {
            sessionStorage.setItem("player2", characterId); // Send player2 id to sessionstorage
            cardCharacterId.style.padding = "12px"; // Padding
            cardCharacterId.style.border = "2px solid blue"; // Border
            changeConfirmId.id = ("confirm"); // Change confirm button id back to "confirm"
            changeDenyId.id = ("deny"); // Change deny button id back to "deny"
            $('#confirmModal').modal('hide'); // Hide modal
            if (sessionStorage.getItem("player1") && sessionStorage.getItem("player2")) {
                $('#startGameModal').modal('show'); // Show start game modal if both players has selected
            }
        }
    } else { // The active player chooses again as "No" is clicked
        console.log(characterId);
        changeConfirmId.id = ("confirm");
        changeDenyId.id = ("deny");
        $('#confirmModal').modal('hide');
    }
}
```

Read about the characters

When the user clicks the “Read more” button, the URL ending of the API link is added, and a fetch with the characterUrl is run. The result, which is converted to JSON from the API is then added to the DOM, and the user can read and learn about the characters. The reason I am using backticks and creating the HTML inside JavaScript is to make the content dynamic with as little code as possible.

```
// If the "read more" button is clicked, add the button id to end of url
function characterInfo_click(clicked) {
    // clicked is the id of the button clicked in the html
    if (clicked) {
        // add id to end of url
        let characterUrl = apiUrl + clicked;
        // fetch said url
        fetch(characterUrl)
            .then((response) => {
                return response.json()
            })
            .then((result) => {
                let character = result;
                document.getElementById("modalBody").innerHTML = `
                <h4 id="characterModalName" class=" [ Cormorant ] "> ${character.name}</h4>
                <p id="characterModalGender" class=" [ Cormorant ] ">Gender: ${character.gender}</p>
                <p id="characterModalCulture" class=" [ Cormorant ] ">Culture: ${character.culture}</p>
                <div>
                    <h5 class=" [ Cormorant ] ">Titles:</h5>
                    <ul id="characterModalTitles" class=" [ Cormorant ] ">
                        ${
                            // learn this by heart
                            // https://stackoverflow.com/questions/45812160/unexpected-comma-using-map
                            character.titles.map(function(value) {
                                return `<li class=" [ Cormorant ] ">${value}</li>`
                            }).join('')
                        }
                    </ul>
                </div>
                <div>
                    <h5 class=" [ Cormorant ] ">Aliases:</h5>
                    <ul id="characterModalAliases" class=" [ Cormorant ] ">
                      ${
                        // learn this by heart
                        // https://stackoverflow.com/questions/45812160/unexpected-comma-using-map
                        character.aliases.map(function(value) {
                          return `<li class=" [ Cormorant ] ">${value}</li>`
                        }).join('') // this part joins the array together as a string and removes the commas between
                      }
                    </ul>
                </div>`
            })
    }
    event.preventDefault();
}
```

When both users have read about the different characters, and chosen a character, the info is sent to sessionStorage, and the “Start Game” modal will be displayed.

The game

I started off by defining the players; player1 and player2. These variables contained an object with the properties tile, id, playerName, and character. The tile property will be changed throughout the game, as every dice roll will affect this value (if the player is active, of course). I also created a variable with a boolean to check if a player is active or not. Since there are only two players at this stage, there is no problem with using a boolean for checking whose turn it is.

```
let player1 = {
    tile: 0,
    id: 1,
    playerName: sessionStorage.getItem('player1'),
    character: document.getElementById("player1img"),
}
let player2 = {
    tile: 0,
    id: 2,
    playerName: sessionStorage.getItem('player2'),
    character: document.getElementById("player2img"),
}

let player1Active = true;
```

To run the game, everything depends on the game function. This function is triggered on the "Roll dice" button in the DOM. Every time the dice is rolled, the active player gets the dice value added to their tile property.

```
function game(){
    // DICE
    let dice = Math.floor(Math.random() * 6) + 1; // Random number between 1 and 6
    // console.log("Dice shows " + dice);
    var diceDOM = document.querySelector('.dice');
    // Spin the dice
    if (diceDOM.classList.contains('rotate')) {
        diceDOM.classList.remove('rotate');
    } else {
        diceDOM.classList.add('rotate');
    }
    // 2. Display the result in DOM
    setTimeout(function () {
        diceDOM.src = 'graphics/img/dice/dice-' + dice + '.png';
    }, 250);


    // give score to correct player
    if(player1Active){
        document.getElementById("playerWho").innerHTML = "Player 2"; // Add current players name in DOM
        player1.tile += dice; // Add the score from dice to player1.tile
        // From tileMovement.js
        player1.tile = checkTrap(player1.tile); // Check if current tile is a trap
        // From tileMovement.js
        checkWinner(player1.tile); // Check if current player is the winner
        // https://www.geeksforgeeks.org/html-dom-appendchild-method/
        document.getElementById(player1.tile.toString()).appendChild(player1.character);

    } else {
        document.getElementById("playerWho").innerHTML = "Player 1";
        player2.tile += dice;
        player2.tile = checkTrap(player2.tile);
        checkWinner(player2.tile);
        document.getElementById(player2.tile.toString()).appendChild(player2.character);
    }

    // From characterTokens.js
    smallerCharactersOnSameTile(); // Make characters smaller if on same tile

    // toggle player if dice is not 6
    if(dice !== 6){
        player1Active = !player1Active;
    }
    // DEBUG
    // console.log(player1.tile + " vs " + player2.tile);
}
```

If player.tile is equal to a trap, the checkTrap function is triggered. In this function, I also decided to go for the switch statement. This allows me to include a case for each trap, so if the player hits the same tile as the case number, the function triggers, and the user is warned with a modal and is sent back as many steps as the trap function returns.

```
function checkTrap(tile){
    switch (tile) {
        case 4:
            showTrapModal();
            trapText.innerHTML = "You just got pushed out of the window. Move back 2 steps";
            // alert( "You just got pushed out of the window. Move back 2 steps")
            return tile - 2;
        case 12:
            showTrapModal();
            trapText.innerHTML = "Khal Drogo just raped you :( 3 steps back";
            return tile - 3;
        case 19:
            showTrapModal();
            trapText.innerHTML = "Hodor is holding the door shut. Move back 5 steps";
            return tile - 5;
        case 26:
            showTrapModal();
            trapText.innerHTML = "Cersei blew up the city... 3 steps back";
            return tile - 3;
        case 32:
            showTrapModal();
            trapText.innerHTML = "You lost a battle.... Move back 21 steps";
            return tile - 21;
        default:
            return tile;
    }
}
```

The rest of the game functions are based on if/else statements, and triggers functions if there is matching information. The reason I mainly use if/else is because I feel that they are correct to use for their purpose. Where there are more things to check, I use switch statements, but this only happens two times throughout this project.

Conclusion

This was a challenging, frustrating, but also a very fun and interesting project. I started off good but got into a serious struggle midway. This gave me a setback, but also the opportunity to do things the correct way without any shortcuts and quick fixes. I am happy with having a working game, but if I had not made a mess in the first place, I think I would have a much better result in both design and functionality throughout the game. I also made some illustrations for the trap modals, that are not included in this report, but you find them in the folder for images.

List of tools/links used during the project

INDEX:

Flame background: https://www.youtube.com/watch?v=LcF6ut-1M94

(Free to use)

SELECTCHARACTER:

Tokens:

- Cartoon effect: https://www.cartoonize.net/
- Photoshop: remove background, add gradient, make black/white, brush tool, make circular

Logo:

- Inspiration for the wings: https://i.ytimg.com/vi/GCagUfe84mQ/maxresdefault.jpg
- Photoshop: tracing, masking, coloring, text

GAME:

Dice

- Sound effect: http://soundbible.com/182-Shake-And-Roll-Dice.html edited in GarageBand
- Illustrator

Trap illustrations

- Illustrator

Winner Modal

- Photoshop: background flames (gif)

Link for design sketches, github and trello

- https://xd.adobe.com/view/4c9f8c96-4d98-4ead-4b64-8db952391c07-3aae/
- https://github.com/bjornivars/GameOfDragons
- https://trello.com/b/zikocRqC/gameofdragons