Arkanoid game documentation

Basic goals and functioning of the game:

- There is 3 different levels you have to complete in order to win the game.
- You have to hit every brick on each level to progress to the next level.
- You win if you complete all levels.
- You lose if the ball hit the ground more than 3 times (lives).
- There is collision logic between the bricks, paddle and walls.
- If the ball hits a brick, the brick disappear.

Description of game controls:

- Use keyboard right and left arrow to move the paddle.

Design of all game screens and menus:

- Three different levels with different amount of bricks.
- Game over screen with a play again button.
- Instruction screen with a back button.
- Main menu screen with start game and instruction buttons.

OOP description of objects in the game (classes, their attributes and methods):

- Node class:

- Implementation of observer pattern to manage relationships between objects in the game.
- constructor(): Creates a new Node object, which initializes an empty array called nodes.
- add(node): Adds a new node to the observer list, by pushing it onto the nodes array. If the argument is false, it returns false.
- remove(node): Removes a node from the observer list, by finding its index in the nodes array and using splice to remove it. If the node is not found, nothing happens.

- notify(event, ...args): Notifies all the observers of a change by calling a method on each of them. The event parameter specifies the name of the method to call, and the ...args parameter is a spread syntax used to pass any additional arguments to the method call.

- **GameObject class:**
  - Base class for all game objects.
  - Attributes:
    - x: x coordinate of the game object.
    - y: y coordinate of the game object.
    - width: width of the game object.
    - height: height of the game object.
  - Methods:
    - draw: draws the game object on the canvas and notifies all children.
    - move: updates the game object's position and notifies all children.

- **Ball class:**
  - extends the gameObject class.
  - Attributes:
    - x: x coordinate of the ball.
    - y: y coordinate of the ball.
    - dx: x coordinate of the ball's velocity.
    - dy: y coordinate of the ball's velocity.
    - radius: radius of the ball.
    - speed: speed of the ball.
  - Methods:
    - ondraw: draws the ball on the canvas.
    - onmove: updates the ball's position and checks for collisions with walls or paddle.
    - reset: resets the ball's position.
    - ballWallCollision: checks for collision between the ball and the walls.
    - ballPaddleCollision: checks for collision between the ball and the paddle.

- **Paddle class:**

- The paddle is also a gameobject.
- Attributes:
  - x: x coordinate of the paddle.
  - y: y coordinate of the paddle.
  - width: width of the paddle.
  - height: height of the paddle.
- Methods:
  - ondraw: draws the paddle on the canvas.
  - onmove: updates the paddle's position based on keyboard input.

- **Brick class:**
  - Attributes:
    - x: x coordinate of the brick.
    - y: y coordinate of the brick.
    - width: width of the brick.
    - height: height of the brick.
    - Row: which row the brick is in.
    - Column: which column the brick is in.
  - Methods:
    - draw: draws the brick on the canvas.
    - update: updates the brick's position.
    - ballBrickCollision: checks for collision between the ball and the brick.

- Game class:

The game class inherits from the GameObject class. It has a constructor that initializes various properties such as the canvas, context, level and score counters, health, bricks array, keys, and time. It also has methods to show the menu, instructions, and to start the game. There is an onloop() method that runs the game loop by updating and drawing the game elements, and it checks for various conditions such as the health of the player, the score, and whether the game is over. It also has a method to create the bricks and methods to create different levels of the game. The game can be controlled using the keyboard arrow keys.

- background class:

- Sets the background to the background image.

- life class:

- draws the lifes and life image on the canvas.

Description of interesting parts of the implementation

The Node class implements the Observer pattern and serves as the base class for the game objects. It allows objects to subscribe and unsubscribe to events and provides methods for notifying subscribed objects of specific events.

I had some problems with the mvc pattern.

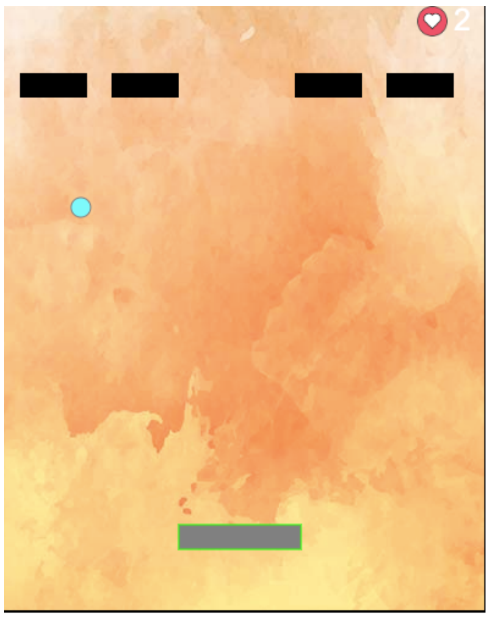
Model: The code does not explicitly define a separate model component. However, the Game class can be considered as the central component that manages the game state, including the level count, score count, health, bricks, and other game objects.

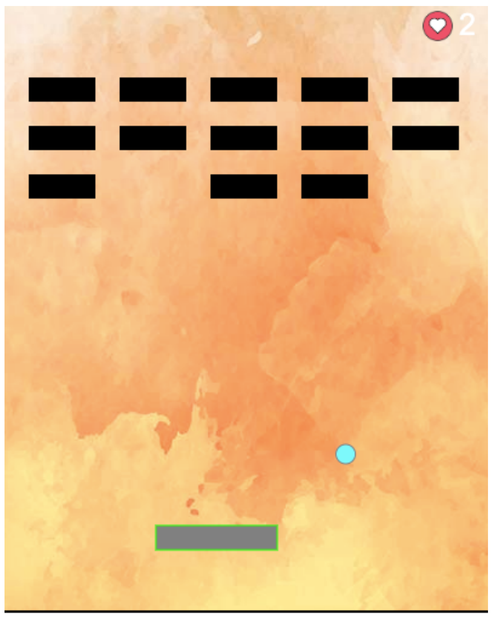
View: The view component is responsible for rendering the game on the canvas and handling user interactions. The Game class manages the canvas and contains methods for drawing various game elements. It also handles user events such as key presses and button clicks.

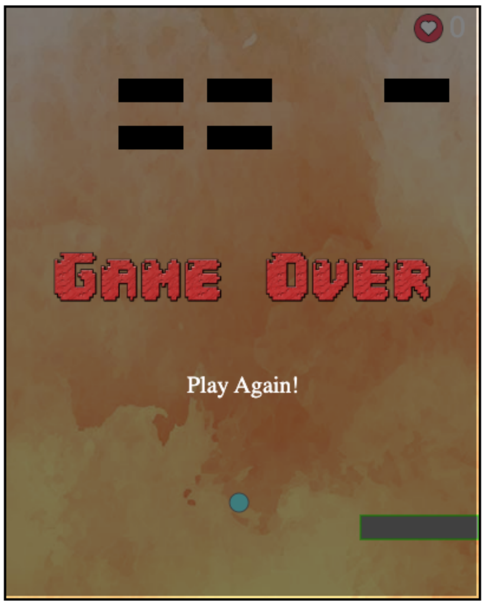
Controller: The code does not have a separate controller component. The game logic, including updating the game state and handling collisions, is directly implemented within the Game class.

While the code does not strictly adhere to the MVC pattern, it does have some separation of concerns by encapsulating related functionality within the Game class and separating rendering from game logic. However, for a more structured MVC implementation, I should probably have separate classes or modules for the model, view, and controller, with clear responsibilities and interactions between them.

Some pictrues from the game:







Graphic and sound elements of the game:

See the file components.js for all the sounds and graphic elements I use in my game.

I have sounds for:

- Ball hitting walls,
- Ball hitting paddle.
- Ball hitting bricks.
- Losing life.
- Winning.

I have image for:

- Background.
- Lifes.
- Game over.
- Winning the game.