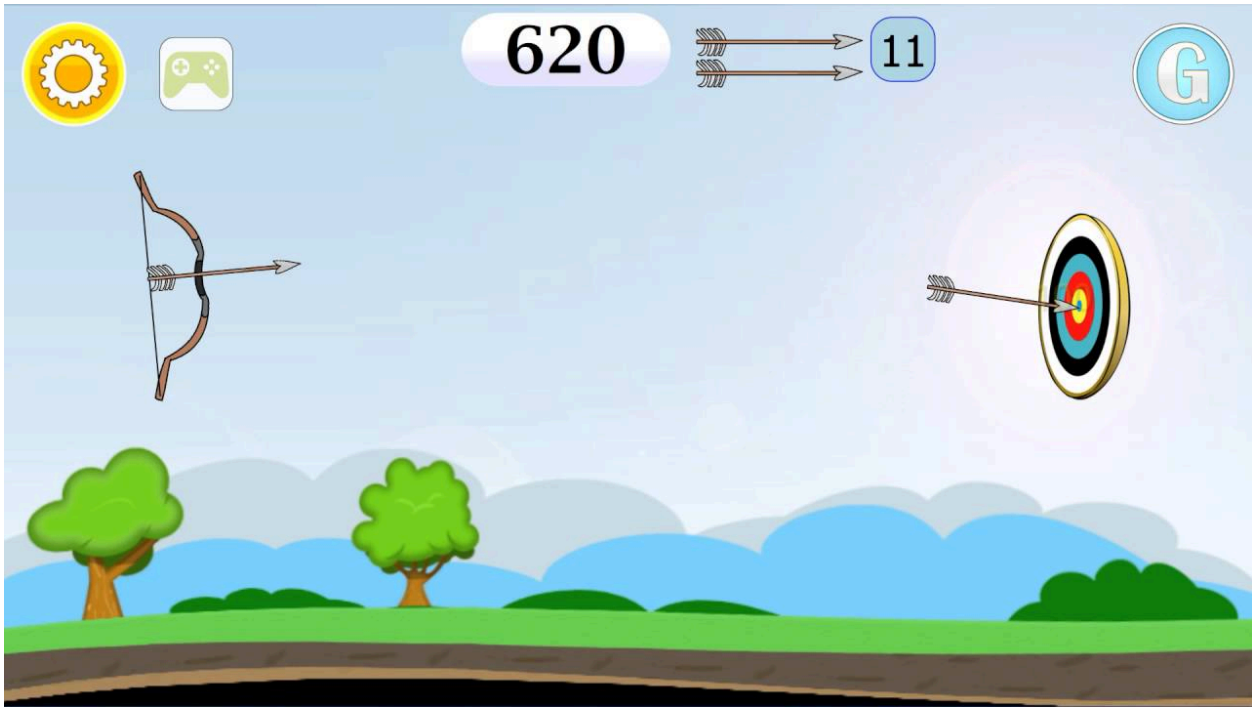
Java Script
Task 2

This archery game is developed using HTML, CSS, and JavaScript where all game elements such as the bow, arrow, and target are created using **div elements**. The player shoots an arrow by clicking or pressing a control, and the arrow moves towards the target using JavaScript logic. The target is not stationary; it moves continuously within a defined area, increasing the difficulty of the game. When the arrow hits the moving target, the score increases. The game uses JavaScript event handling and DOM manipulation to control movement, collision detection, and score updates.



Criteria	Updated Description (with coding changes)	Marks
DOM Element & Object Usage	Correctly selects DOM elements using modern JavaScript methods (`querySelector`, `getElementById`). Uses variables/constants appropriately and avoids hard-coding values. Elements are created or updated dynamically where needed.	1
Event Handling Logic	Implements event listeners efficiently using `addEventListener`. Correctly distinguishes between target clicks and background clicks using event objects. Prevents unintended behavior such as event bubbling when required.	1

Function & Code Structure	Code is modular and readable with well-named functions for movement, timer, score update, and difficulty control. Uses arrow functions or standard functions consistently and avoids duplicated logic. You create **separate functions** for different tasks, for example: • `moveTarget()` → moves the archery target • `startTimer()` → starts and updates the countdown • `updateScore()` → increases score when the target is hit • `increaseDifficulty()` → makes the game harder Function names clearly describe what they do. You **do not repeat the same code** in multiple places. You use **one style of functions consistently**: • Either arrow functions `() => {}` • Or normal functions `function name() {}`	3
Game Logic & State	Maintains clear game state variables (score, time, game status). Correctly handles timer countdown, game over condition, score updates, and reset/restart logic without bugs. You use variables to track the game state, such as: • `score` → number of targets hit • `timeLeft` → remaining game time • `gameOver` → whether the game has ended	3

	The **timer counts down correctly** every second. When time reaches zero: ● The game stops ● Target movement stops ● A **game over message** is shown The **score only increases** when the player hits the target. The game can be **reset or restarted** without errors.	
Dynamic Styling & Difficulty	Dynamically updates styles using JavaScript (size, speed, color, position). Difficulty increases logically based on score or time (e.g., faster movement, smaller target). Changes are smooth and responsive.	2
Total		**10**

Extra Help:

This archery game is developed using HTML, CSS, and JavaScript where all game elements such as the bow, arrow, and target are created using **div elements**. The player shoots an arrow by clicking or pressing a control, and the arrow moves towards the target using JavaScript logic. The target is not stationary; it moves continuously within a defined area, increasing the difficulty of the game. When the arrow hits the moving target, the score increases. The game uses JavaScript event handling and DOM manipulation to control movement, collision detection, and score updates.