TEACHER NOTES - COOKIE CLICKER

Adam Chapman-Dodd and Christopher Dalziel

April 16, 2025

Lesson guide

For this lesson you should have, and be familiar with the below materials:

- Workbook
- Teacher notes (this document)
- PowerPoint presentation
- HTML, CSS and JS template files and model solution files

The intention of this lesson is to help higher computer science students gather an understanding on Event-driven programming in JavaScript through 'making' a game.

The lesson follows a **Learn-Do-Discuss** methodolgy in which the students will take some time to discover and **learn** about a topic (through the presentation slides), **do** a task to help bridge that theoretical understanding to something visual and tangible (tasks in the workbook fixing/creating features in the game), and then **discuss** the concept, how it was implemented in the game and tie both the theoretical approach and practical solution together.

This discussion is vital in tying the concepts together and the students should be encouraged to *think* about it after they have implemented a feature, talk about this with their classmate (*pair*) and then, as in the presentation, *share* the thoughts with the class - the *think-pair-share* approach.

Hints are provided for both the tasks and discussions if students are struggling with either part. The discussion hints may be especially helpful to begin the full-class conversation if students are not sure or not wanting to start.

Hints, Solutions and guidance for Discussion for each task:

1. The student should complete the function `cookieClick`.

   - This function should increase the `score` value by the `clickPower` (or by 1 if the student is not yet considering upgrades)
   - The `clickedLastSecond` should also be increased by the `clickPower`. This however is not relevant until task 2 (see below).
   - The student will need to update the display by calling `refreshCookieCount` at the end of the function. If struggling to see this step, they should be encouraged to spend some time reading through the pre-written function and testing calling it in their code.

     ```
     function cookieClick() {
         score += clickPower;
         clickedLastSecond += clickPower;
         refreshCookieCount();
     }
     cookie.addEventListener('click', cookieClick);
     ```

   - If the student wished to complete the bonus task here, they are expected to edit the `css` file. They can be given a hint about using `#cookie:active` for a change in size and other visual aids, with an example of how to do so in the below solution

     ```
     #cookie:active {
         transform: scale(0.9);
         box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.2);
     }
     ```

□ Discussion questions for **TASK 1**:

- Why do we separate logic (JS), styling (CSS) and structure (HTML) rather than keep it all in the same file?
  - *Hint:* Think about where your JavaScript code is compared to your HTML layout. What difficulties could you face if everything was written in one place?
  - **Example Answer:** Separating these makes the code more readable, easier to maintain, and more reusable. HTML handles content and structure, CSS deals with appearance, and JavaScript controls behaviour. Keeping them apart means different parts can be worked on without breaking others (which makes group programming more manageable).
- How might you use different types of events (like `mouseenter`, `dblclick` or `keydown`) to change how players interact with this game?
  - *Hint:* What other actions can users perform other than just clicking the cookie? Think about games or apps you've used - what inputs do what?
  - **Example Answer:** You could use `keydown` to let users press a key to click the cookie instead, or `mouseenter` to show a tooltip or animation when hovering over the cookie.
- Why is `refreshCookieCount()` called after increasing the score? What would happen if we forgot to call it?
  - *Hint:* `refreshCookieCount()` sets the score/cookies displayed to the value in the code.
  - **Example Answer:** `refreshCookieCount()` updates the visible score on the page. If we increase the score in the code but don't call this function, the user won't see the change and it would look as though nothing happened (even though the value actually increased).
- How could you make this click interaction more engaging or apparent for a user?
  - *Hint:* In games you have played or websites you have used, what other feedback is used to signal that you have interacted with an object?
  - **Example Answer:** You could add further animations, such as a smaller cookie firing off on click, to make the cookie button more visually responsive to clicks. Adding sound effects could also make the interaction more noticeable.

2. The student should fix the CPS display by adding code into the empty `setInterval` call.

   - If not added previously, the student should be encouraged to revisit their `cookieClick` function and think about how clicking would affect the `clickedLastSecond` variable. See above notes (task 1) for more details.
   - The student should recognise the `updateCPS` function written above and see to call this with `clickedLastSecond` as an input.
   - The student should also discover they need to set `clickedLastSecond` to 0 at the end of each second interval (otherwise it will just keep increasing any time cookies are scored).

```
setInterval(() => {
    updateCPS(clickedLastSecond);
    clickedLastSecond = 0; // Reset
}, 1000);
```

   □ Discussion questions for **TASK 2**:

   – Why use `setInterval()` instead of updating CPS on each click? What problems could occur if CPS was updated on every frame?

   * *Hint:* Think about how often updates are needed and whether it's practical to update CPS every time a user clicks. Could this result in the feature not working as expected, or other issues?

   * **Example Answer:** `setInterval()` allows us to update the CPS value every second, which is more efficient than updating it every frame as it reduces unnecessary calculations. It also will provide a smoother and less 'jumpy' value than if updated every frame (due to different things adding to the CPS). If we updated only every click, then we could not account for things like auto-clicking upgrades.

   – Why do we need to reset the counter each second? What happens if you simply don't do this?

   * *Hint:* Try removing this from the code. Think about how the CPS value should be calculated and why the cookies count from the previous second should not carry over.

   * **Example Answer:** The counter must be reset each second so that the CPS meter shows only the cookies clicked during the past second. If the counter isn't reset, it would accumulate the cookies earned over time, resulting in a misleading and incorrect value that grows constantly.

3. The student should fix and create all necessary upgrades by changing values and looking at console error messages.

   - If not added previously, the student should use a look through the values of upgrade 1 and recognise that it is not set up to change. `scalingAmount` is 1 and so the cost will not change. `power` and `clickPower` in the `onUpgrade` function are also both 1: these should increase with the level. If the student is unsure on where to start with fixing this, they should be encouraged to first check that the button works (and the code is being reached) with `console.log`

```
createUpgrade({
    buttonId: 'buy-click-power',
    priceElementId: 'click-power-price',
    levelElementId: 'click-power-level',
    multipleElementId: 'click-power-multiple',
    priceAmount: 10,
    scalingAmount: 2.33,
    levelNumber: 1,
    power: (level) => level,
    onUpgrade: function(level) {
        clickPower = level;
    }
});
```

   - The student should add their code into the `setInterval` to add the auto-clicker value(s) to the score each second. They, again, need to make sure that this also updates the display. They must also find the spelling error that causes the first auto-clicker (upgrade 2) to not work (they should be pushed to use console debugging to see this issue).

```
createUpgrade({
    buttonId: 'buy-auto-clicker-1',
    priceElementId: 'auto-clicker-price-1',
    levelElementId: 'auto-clicker-level-1',
    multipleElementId: 'auto-clicker-multiple-1',
    priceAmount: 100,
    scalingAmount: 2.5,
    levelNumber: 0,
    power: (level) => level,
    onUpgrade: function(level) {
        autoClickerPower1 = level;
    }
});
```

```
setInterval(() => {
    // Add auto-clicker power to score each second
    score += autoClickerPower1 + autoClickerPower2;
    refreshCookieCount();
}, 1000);
```

- The student should then explore this `createUpgrade` function by adding their own autoclicker upgrade. They should use `autoClickerPower2` and update previous functions to account for this. They must also add in the necessary HTML elements

```
createUpgrade({
    buttonId: 'buy-auto-clicker-2',
    priceElementId: 'auto-clicker-price-2',
    levelElementId: 'auto-clicker-level-2',
    multipleElementId: 'auto-clicker-multiple-2',
    priceAmount: 200,
    scalingAmount: 2.5,
    levelNumber: 0,
    power: (level) => (level*2),
    onUpgrade: function(level) {
        autoClickerPower2 = (level*2);
    } });
```

```
<div class="shop-elem">
    <p>Auto Clicker 2</p>
    <p>Produces
        <span id="auto-clicker-multiple-2">0</span>
    cookie(s) per second </p>
    <button id="buy-auto-clicker-2">Buy for
        <span id="auto-clicker-price-2">200</span>
    </button>
    <p>level: <span id="auto-clicker-level-2">0</span> </p>
</div>
```

□ Discussion questions for **TASK 3**:

- Why is it a good idea to have one function like `createUpgrade()` handle upgrade setup?
  - *Hint:* Think about functions as a whole and how a change to the upgrade logic might affect multiple upgrades.
  - **Example Answer:** Having `createUpgrade()` manage upgrade setup keeps all the upgrade logic together, making the code more modular and easier to maintain and expand. If we need to change how the upgrades work or add new upgrades (like in this task), we can use or modify this function accordingly (rather than do everything manually or editing each upgrade individually).
- What is the issue with upgrade prices not scaling with level?
  - *Hint:* What happens if you set the upgrade price to not scale with the level? Can you see how this may 'break' the game or be exploited?
  - **Example Answer:** If upgrade prices don't scale with level, the upgrades might become too cheap or stay too expensive. We don't want this unbalanced progression and scaling prices helps to keep the game challenging and rewarding as the player advances.
- How do HTML element IDs help JavaScript interact with the correct parts of a web page? What would happen if IDs are reused?
  - *Hint:* Think about how JavaScript finds HTML elements and the role of IDs in identifying them.
  - **Example Answer:** HTML element IDs allow JavaScript to find and change specific elements on the page. If IDs are reused, JavaScript might interact/update with the wrong element or cause another error (`getElementById()` expects a unique identifier).

4. The student should fix the golden cookie functionality by implementing randomised values for the variables used in the creation of the golden cookies.

   - `cookieVal` and `delay` should just be any value written using a `Math.random()` function. There is no 'right answer' for this, however the student should be encouraged to play around with values to see what feels natural and correct here.

```
// Random number 1 - 101 (weighted to lower values)
    const cookieVal = Math.ceil(Math.sqrt(Math.random() * 10000) + 1);
    // 20s to 40s
    const delay = Math.random() * 20000 + 20000;
```

   - The `duration` should be any value that changes with the `cookieVal` value. Again, there is no correct answer but instead the student should test or try and provide some justification for their choice of duration (e.g. more valuable cookies should be more difficult to click and therefore appear for less time).

```
// random duration based on value
    const duration = (20000-((cookieVal + 30)**2))/2;
```

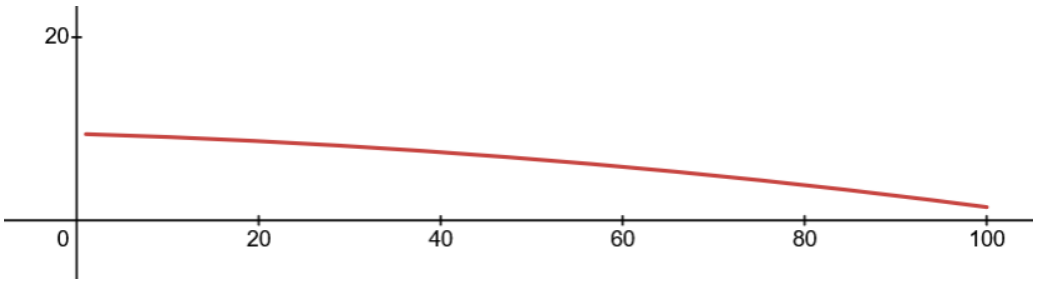

Figure 1: A graph showing the golden cookie duration decreasing for the more expensive cookies with the formula used in the solution

   - The student should recognise, from the previous tasks, that there is a missing call to the `refreshCookieCount` function in the golden cookie event listener (the error from not including this may be seen by the student clicking the golden cookies and nothing happening straight away). Finally, there will be no golden cookies unless the student adds in a line to start the cycle of spawning, there is a hint to this in the template.

```
spawnGoldenCookie(); // Start the loop
```

   - For the chips, it is simply a case of copying the format of the HTML and CSS for the cookie, changing the size, colour and margins and placing it inside the golden cookie `<div>`.

□ Discussion questions for **TASK 4**:

- Why do we use `Math.random()` here? How could you skew that to be weighted towards lower numbers?
  - *Hint:* `Math.random()` returns a random value $0 \leftrightarrow 1$. Think about how you could change the distribution by using mathematical operations (like `Math.sqrt()`), or even drawing a graph.
  - **Example Answer:** `Math.random()` gives a uniform distribution, so, if we want to make lower values more likely, we can use something like `Math.sqrt(Math.random())`, which biases results toward smaller numbers. Having randomness in these components adds variety to the game and keeps it engaging. If players always knew exactly when and where a golden cookie was spawning, it would feel repetitive and not have any sort of challenge.
- Why do we use `setTimeout()` instead of `setInterval()` for controlling how long a golden cookie stays on screen?
  - *Hint:* Do we want the golden cookies to be removed after a certain period of time, or at regular intervals?
  - **Example Answer:** `setTimeout()` is used when we want something to happen after a specific delay (e.g. removing the golden cookie after a certain time). In this instance, `setInterval()` would keep trying to remove the golden cookie from the screen, which doesn't make sense and so the `setTimeout()` is much more suitable.
- How could we make golden cookies rarer or more powerful? How would you start implementing that?
  - *Hint:* What if, instead of giving an amount of cookies, it doubles your CPS for the next minute: how would you start implementing that?
  - **Example Answer:** We could make golden cookies give different effects (e.g. see *hint*). To implement this, we could use `setTimeout()` to activate an effect and then eventually reset it after the timer finishes. We would have to find a way on calculating the original and new CPS to go to and from, as well as finding a way to show this effect is active (visually and/or audibly).

Extension

Bonus Features - Hints of how to implement

- Add sound effects - Use JavaScript's `Audio` object to play a sound when a cookie is clicked or an upgrade is bought (`https://noaheakin.medium.com/adding-sound-to-your-js-web-app-f6a0ca728984`).
- Add achievements and new upgrade types - Track conditions like number of clicks or total cookies with a variable. When thresholds are met, display a message or change the level/price of a new upgrade.
- Save game progress using local storage - Check that `localStorage` is supported and use `setItem` and `getItem` to store and retrieve key-value pairs to store progress locally (`https://gamedevjs.com/articles/using-local-storage-for-high-scores-and-game-progress/`).

Further Discussion Prompts

- What part of this project did you find the most difficult? *Is there something you could now add or improve in your game that would show you've overcome that challenge?*
- Which coding techniques did you think is most useful or reusable? *Can you think of ways you might apply them in other projects or programming languages?*
- If you had more time to explore, test and build, what features would you add? *How would you make a start in adding these?*