JavaScript 網頁程式設計

期末專題

Triple Town 益智遊戲

系級：應用數學系四年級

姓名：林家生

學號：0712239

第一部分：玩家角度

遊戲名稱：

Triple Town 益智遊戲

遊戲包含檔案：

- Project-0712239.html
- Project-0712239.css
- Project-0712239.js
- 物品圖片共 12 張 (.png 檔)

遊戲開啟方式：

將上述檔案放在同一資料夾路徑，並用瀏覽器開啟 Project-0712239.html 檔案，即可開始遊戲。***小提醒，瀏覽器下方若有下載列要記得關閉，否則遊戲畫面會擠壓到！***

遊戲介紹：

這是一款在網頁上的單機類益智遊戲，玩法模仿多年前上架的手遊“Triple Town”，玩家要在抵禦兔子大軍襲擊的同時，透過相同物品組合升級來盡可能獲取分數，不僅考驗城市規劃能力，更考驗臨場反應。

遊戲目標：

在面積有限的土地裡，找尋屬於自己的遊戲策略，想盡辦法拿到更多分數，突破自己吧！

遊戲中登場的物件：

- 建築物件：小草、灌木叢、大樹、小屋、城堡。
- 兔子物件：普通兔、反派兔。
- 分數物件：一根胡蘿蔔、一箱胡蘿蔔、錢幣。
- 特殊物件：水晶、槌子。

各個物件皆是用小畫家畫出來的 (累...

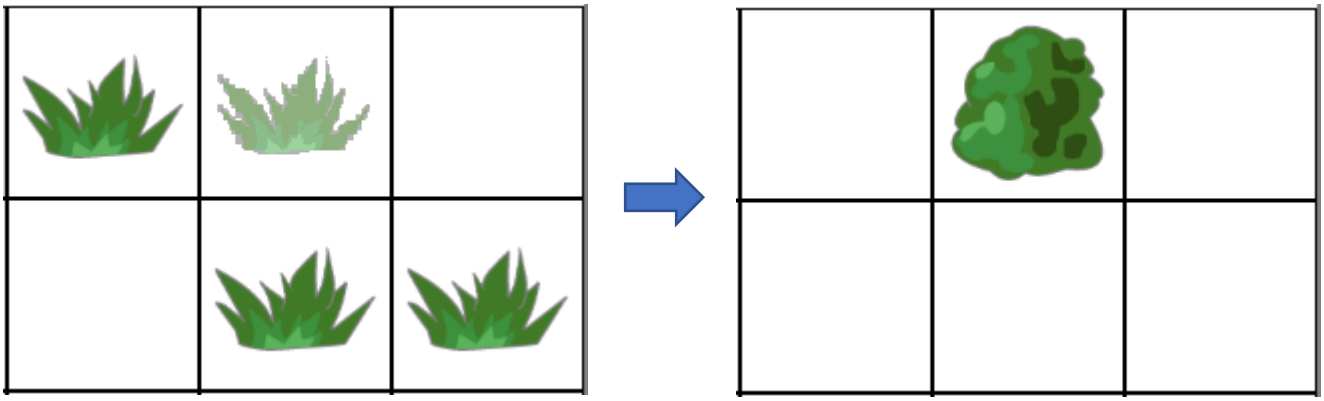
有經過去背處理，背景為透明。

(其中藍色箭頭表示 ≧ 3 個相鄰即可升級)

(紅色加分表示合成/點擊該物件可獲取相應分數)

遊戲介面及玩法詳細介紹：(有點冗長，可以邊玩邊參考)

- 進到遊戲後，上方 Current Item 會顯示下個出現的物件，可透過滑鼠點擊的方式將物品擺放到下方空格上 (預設共有 6 * 6 = 36 格)。
- **若有出現大於等於 3 個相同物件相鄰的狀況 (斜角不算)，最後放下的物件將會升級成為下一階物件，其餘相鄰同種物件會消失，且玩家將會獲取一定的分數。**



- 上方 Hold 按鈕可以將當前的 Current Item 暫時保存，最多一個。
- 普通兔和反派兔也有概率出現在物件序列中，且擺放在下方空格同樣會占用一格欄位，活潑亂跳的他們可能會影響玩家原有的建築規劃喔！
  - **場上的普通兔每回合都有機會朝上下左右移動一格，若放下物件時，兔子四個方位都被建築物件或分數物件所包圍，則會逃跑並留下一根胡蘿蔔。**

- 場上的反派兔則是更為討厭，每回合都有概率會跳往當前場上的空白格，若場上被塞滿了沒有地方跳，則他們會全部逃跑並留下一箱胡蘿蔔。

- 一根胡蘿蔔則是屬於比較特別的分數物件，和一箱胡蘿蔔、錢幣同理，雖然在場上佔用空格，但玩家可在任一時間點擊分數物件以清空該空格，同時玩家也會獲得定量的分數。(這邊要提醒的是，三者物件的分數並不成線性比例，因此要選擇往上升級還是立即換取分數則由玩家權衡取捨囉！)
- 特殊物件則分為兩種：水晶與槌子。
  - 特殊物件並不能直接被擺放在場上，會以特殊形式和其他物件互動。
  - 水晶能和各類物件組合湊數量升級，若擺放位置同時有兩組符合條件，則會依照合成順序高到低做選擇，順位如下：一箱胡蘿蔔 > 一根胡蘿蔔 > 小屋 > 大樹 > 灌木叢 > 小草。
  - 槌子可以破壞掉場上的物品，讓擺放位置不佳的物件能被清除。
  - 槌子也可以將兔子物件打跑，並掉落相對應的胡蘿蔔。
  - 若水晶和槌子被擺放在空白處 (且水晶周圍沒有能與之合成的物品)，則他們會變成一根胡蘿蔔。

第二部分：設計角度

開發環境：Windows 10，版本 21H2

開發平台：Visual Studio Code (version: 1.67)

瀏覽器：Chrome 版本 102.0.5005.63 (正式版本) (64 位元)

為什麼選擇這個題目：

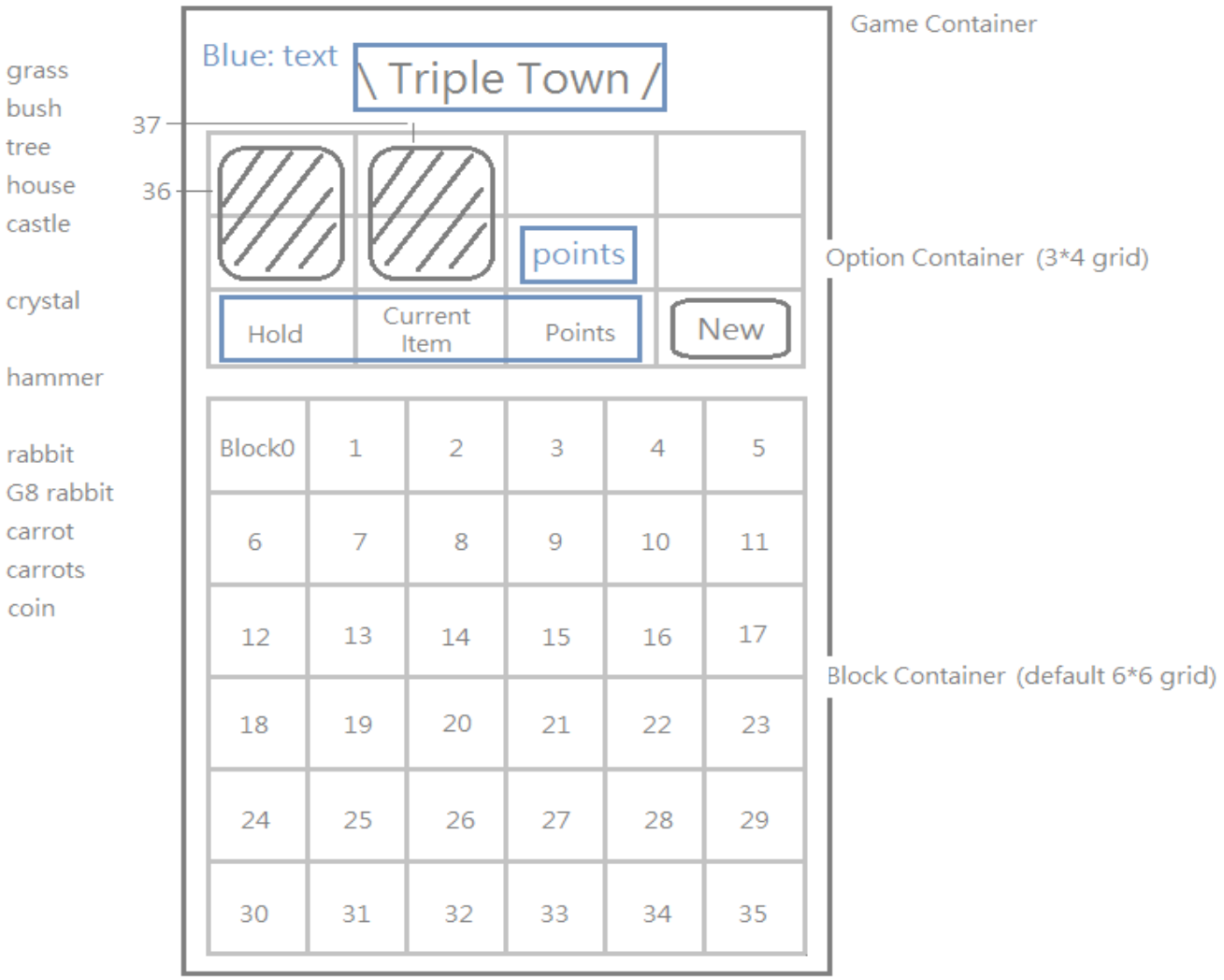
其實我原本有考慮的題目有踩地雷、桌上曲棍球、停車遊戲、美食地圖等，但最終因為怕跟其他人撞題目，且希望創新，所以就選了 Triple Town (譯：三合一城鎮)這個感覺比較少人知道的手機益智遊戲。

印象中我初次接觸到這款遊戲大概是在國中二三年級的時候，玩過的人好像不多，但當時我對於它的遊戲想法就滿驚嘆的，雖然動作只有思考並點擊空格擺放物品，但他的各種設計以及向玩家呈現的方式都令人印象深刻，也因此，這次借助期末專題機會，我決定用網頁版的方式復刻這款遊戲，也希望運用這學期課程中所學到的，加入一些自己的想法或是新的元素，讓其他人認識、了解這款遊戲。

設計理念：

這題目某程度來說比較簡單，但看似簡單的背後，卻也隱藏了眾多細節與連動。老實說我在過程中確實遇到了許多麻煩，有時也會想說這邊要不直接改遊戲規則就好，或是一些玩家不太可能被觸發的 bug 就先不理它了吧.....等，但隨著遊戲慢慢成形，我也逐漸認知到不能只把眼光放在程式碼上，遊戲本身也很重要，玩法是否保持合理性，哪些地方是必須堅持的底線，設計難度和趣味性該怎麼兼顧等等的，我一直在這方面做努力。我希望能找到那些平衡點。

最終版草稿圖：



1. 由最外層 Game Container 包住 Option 和 Block，皆是 div 標籤。
2. 上方藍色框住部分是固定的文字顯示，藍色 Points 部分是用 div 標籤框住，會隨著分數增加改變節點 innerText 內容。
3. 圓框框代表是用 button 標籤實作的：
   - Hold 按下會和 Current Item 做交換。
   - Current Item 用途是顯示當前要擺下的物品，按下不會有反應。

     之所以不用 div 而是用 button 做，是因為照片貼上時的顯示效果較佳。
   - New Game 按下會開啟新的一局，是唯一刷新遊戲的方式。(網頁重新整理不會損失前次遊戲進度，靈感來自 wordle 一天只有一題，重整不刷新)

4. Block 部分有一個 Class，在遊戲初始化時會用建構子產生目標數量的 div 格子並存進全域 Blocks 陣列中。

   在程式碼中，主要是用 blocks[index]取用各區塊，用 blocks[index].item 辨別該格當前的物品狀態。

```
class Block{
    constructor(index){
        this.node = document.createElement("div");
        this.node.id = `block${index}`;
        this.node.className = "block";
        this.index = index;
        this.item = "none"; // default item: "none"
    }
```

遊戲方式實作：(目前我的寫法是類似物件導向的感覺，用函數去管理每個操作，以下說明會盡量用適當的順序去陳述。)

遊戲主要操作就是將當前物品，透過點擊的方式，擺放到畫面的空格處。在每一次的空格點擊中，都會觸發該空格的成員函數 changeTypeByClick()，這是整個程式碼的核心軀幹，主導了整個遊戲的進行：

一、 根據該「空格擺放前的狀態」及「即將擺放的物品」，會有不同的篩選過程，且同時有機會觸發合成升級的動作。注意，若此部分判斷當次點擊是無效的點擊，則後面步驟將不會繼續執行。

```
/* 1. Put the item down and check upgrading */
if(this.item == "none"){···
}

/* 2. Items -> none by a hammer */
else if(itemPool[currImgIndex] == 'HAMMER'){···
}

/* 3. Carrots and coin -> points */
else if(this.item == "CARROT" || this.item == "CARROTS" || this.item == "COIN"){···
}

/* If this click has no impact, return */
else return;
```

1. 若點選的空格是空的：

   - 建築物件會檢查是否能合成升級(可能沒有、一次、連鎖兩次)。

```
switch(itemPool[currImgIndex]){ // e.g. "GRASS" (type: string)
    case "GRASS":
    case "BUSH":
    case "TREE":
    case "HOUSE":
        //May combine and upgrade twice successively
        this.item = itemPool[currImgIndex];
        while(this.bfsForCombination(this.item)){
            this.upgrade();
        }
        this.updateImage(this.item);
        break;
```

   - 兔子不會升級，但是要更新「出現到場上的順序」，這在之後會說明。

```
case "RABBIT":
    rabbitOrder.push(this.index);
    this.updateImage(itemPool[currImgIndex]);
    break;

case "G8RABBIT":
    G8rabbitOrder.push(this.index);
    this.updateImage(itemPool[currImgIndex]);
    break;
```

   - 特殊物件放空格會變一根胡蘿蔔，水晶會依優先序檢查能否與周圍合成。

```
case "CRYSTAL":
    let isUpgraded = false;
    for(let item of CrystalCombinePriorty){
        isUpgraded = this.bfsForCombination(item);
        if(isUpgraded){
            this.item = item;
            this.upgrade();
            break;
        }
    }
    if(!isUpgraded) this.item = "CARROT";
    this.updateImage(this.item);
    break;

case "HAMMER":
    this.updateImage("CARROT");
    break;
```

```
const CrystalCombinePriorty = ["CARROTS", "CARROT", "HOUSE", "TREE", "BUSH", "GRASS"];
```

2. 若點選的空格不是空的，且即將擺放的物品是槌子：

   - 點選兔子會變胡蘿蔔，記得維持兔子順序。
   - 點選其他會清空。

```
/* 2. Items -> none by a hammer */
else if(itemPool[currImgIndex] == 'HAMMER'){
    switch(this.item){
        case "RABBIT":
            this.updateImage("CARROT");
            maintainOrder("RABBIT", this.index, -1);
            break;

        case "G8RABBIT":
            this.updateImage("CARROTS");
            maintainOrder("G8RABBIT", this.index, -1);
            break;

        default:
            this.updateImage("none");
            break;
    }
}
```

3. 若單獨點選分數物件：

   - addPoints()獲得分數，UpdateImage(“none”)清空物件。
   - 這階段做完會直接跳出，因為點擊胡蘿蔔、錢幣等物品不會觸發兔子移動，也不會更新當前要擺下物品，更不會導致遊戲結束。
   - 但還是要記得用 setLocalStorage()紀錄遊戲進度。

```
/* 3. Carrots and coin -> points */
else if(this.item == "CARROT" || this.item == "CARROTS" || this.item == "COIN"){
    this.addPoints(this.item);
    this.updateImage("none");
    setLocalStorage(); // store game information to the window.localstorage
    return;
}
```

二、 在各格子物品狀態確認後，會判斷遊戲是否結束，意思也就是，是否已經無法再擺放下個物品。

- 首先會先計算當前場上各物品數

```
/* Count items */
let g8r = [];
let r = [];
let carrot = [];
let carrots = [];
let coin = [];
let n = [];
for(let i = 0; i < BlockNumber; i++){
    if(blocks[i].item == "G8RABBIT") g8r.push(i);
    else if(blocks[i].item == "RABBIT") r.push(i);
    else if(blocks[i].item == "CARROT") carrot.push(i);
    else if(blocks[i].item == "CARROTS") carrots.push(i);
    else if(blocks[i].item == "COIN") coin.push(i);
    else if(blocks[i].item == "none") n.push(i);
}
```

- 若 Current Item 跟 Hold Item 都沒有槌子，且場上已經沒有分數物品，又剛好場上沒有空格擺放物件了，則遊戲結束，並顯示分數。

```
/* 4. Game over or not */
let isG8rabbitEscape = false;
if(n.length == 0){
    isG8rabbitEscape = true;
    if(holdItem != "HAMMER" && itemPool[currImgIndex + 1] != "HAMMER"){
        if(r.length == 0 && g8r.length == 0 && carrot.length == 0 && carrots.length == 0 && coin.length == 0){
            alert(`Game over!\nTotal points: ${points}!`);
            return;
        }
    }
}
```

三、 若遊戲尚未結束，則場上的普通兔和反派兔有機率會逃跑或移動。兔子逃跑後留下的胡蘿蔔也需要檢查是否會觸發合成升級。

- 用兩個函數區分反派兔和普通兔，反派兔會先移動。
- 函數詳細說明放在後面。

```
/* 5. Rabbits may escape or move to another block */
this.g8rabbitEscapeOrMove(isG8rabbitEscape, g8r, n);
this.rabbitEscapeAndMove(r);
```

四、 準備下個物品，並替換當前物品圖片。

- currImgIndex 初始化為 0，每當經過一次有效點擊都會+1，itemPool 裡放的是物品種類的字串，以此來更新當前物品。
- itemPool 長度為 100，所以索引值超過時會有刷新 itemPool 的動作。

```
/* 6. Prepare next item */
currImgIndex++;
if(currImgIndex == 100) resetItemPool();
updateImageOfCurrItem();
```

五、 將資訊存進 window.LocalStorage 中，這是為了將來關掉瀏覽器再打開時，遊戲進度不會消失。

- setLocalStorage()詳細說明同樣放在後面。

```
/* 7. Store game information to the window.localstorage */
setLocalStorage();
```

程式詳細說明目錄： (以下會分成三部分)(截至 111/6/7)(仍待補充)

實作大致流程：

- 遊戲框架設計 (多個 div 要怎麼顯示、位置、大小調整)
- 點擊空格後跳出物件圖片 (圖片怎麼加載、更換等)
- 每個物件有不同的出現機率，該怎麼實現？
- Hold 功能的實作 (與 Current Item 對調)
- 分數的計算及顯示

實作過程中會遇到的困難點：

- 如何判斷該物件擺放時需升級？

- 連鎖升級的問題
- 普通兔的移動 & 反派兔的移動
- 如何判斷兔子是否需要逃跑
- 兔子逃跑、移動時的淡入淡出動畫
- 兔子逃跑後產生的胡蘿蔔合成問題

想新增的一些特色：(6/25 前有完成的用紅色標記)

- 遊戲介面改善 (例如：左右資訊說明欄、hover 顏色改變)
- 紀錄遊玩進度，關掉視窗下次打開可以繼續玩，直到按下 New Game 才會換新的一局
- 剛進遊戲場上會有一些物品
- 玩家可調整遊戲難度，色調主題
- 放到網路上大家都可以玩 (例如：放到 github 上)

  (連結：https://github.com/JasonLin0704/Triple-Rabbit)
- 背景音、按鍵音
- 彈出式商城 (可用分數購買各項物件、倒退或移動物品一次的機會等)
- 透過外部 JSON 檔案自訂義變數，如：格子數、合成分數等
- 增加網頁 icon

程式詳細說明：

✧ 遊戲框架設計 (多個 div 要怎麼顯示、位置、大小調整)

- 三個 Container 形成介面，Game 和 Option 位置是用 relative 決定。
- Block 部分我希望能自訂義格子數，因此是透過 JavaScript 動態產生的，擺放位置是用 absolute。

```
<body>
    <div id="gameContainer">
        <h1 id="title">\ Triple Town /</h1>
        <div id="optionContainer">
            <!-- Hold -->
            <button type="button" class="option optionFirstRow" id="holdButton">click me</button>
            <div class="option optionSecondRow" id="holdButtonText">Hold</div>
            <!-- Next Item -->
            <button type="button" class="option optionFirstRow" id="currItemButton"></button>
            <div class="option optionSecondRow" id="currItemText">Current Item</div>
            <!-- Points -->
            <div class="option optionFirstRow" id="points"></div>
            <div class="option optionSecondRow" id="pointsText">Points</div>
            <!-- New Game -->
            <button type="button" class="option optionFirstRow" id="newGameButton">New Game</button>
            <!-- <div class="option optionSecondRow" id="newGameText"></div> -->
        </div>
        <script src="Project-0712239.js"></script> <!-- JavaScript link -->
    </div>
</body>
```

- 我希望 Block Container 寬度和外層 Game Container 寬度一致。

  因為 block Container 為正方形，因此先決定寬度後，再去找 top 的位置。

```
/* Create the Block Container */
function createBlockContainer(){
    blockContainer = document.createElement("div");
    gameContainer.appendChild(blockContainer);
}

/* Set properties of the block container */
function setPropertyOfBlockContainer(){
    blockContainer.id = "blockContainer";
    blockContainer.style.height = BlockContainerHeight + "px";
    blockContainer.style.width = BlockContainerWidth + "px";
    blockContainer.style.top = GameContainerHeight - BlockContainerHeight + "px";
    blockContainer.style.gridTemplateColumns = `repeat(${BlockColumnNumber}, 1fr)`;
    blockContainer.style.gridTemplateRows = `repeat(${BlockRowNumber}, 1fr)`;
}
```

- 其中值得一提的是，我希望能依照玩家螢幕的尺寸去改變我的框框大小，因此我會讀取電腦解析度當作我的 Game Container 大小的參考。

```
const WindowHeight = window.innerHeight;
const WindowWidth = window.innerWidth;

const GameContainerHeight = WindowHeight * 0.95;
const GameContainerWidth = WindowWidth * 0.3;
```

✧ 點擊空格後跳出物件圖片 (圖片怎麼加載、更換等)

- 我的做法是先將圖片載入並全部存進陣列中，並在需要顯示的時候將圖片節點 append 到該格子 div 節點下，並確保任何時間 div 下只會有一個 child。
- 由於 36 格空格(預設 6*6)以及 Hold 跟 Current 都有可能同時出現同一張照片，所以不能只用單一變數存取，而是要有 12 種物品*38 格，共 456 個 img 標籤節點要初始化時就先準備好。 (但這應該是個很差的做法> <)

```
const ItemName = [
    "GRASS", "BUSH", "TREE", "HOUSE", "CASTLE",
    "RABBIT", "G8RABBIT", "CARROT", "CARROTS",
    "COIN", "CRYSTAL", "HAMMER"
];
```

```
/* Load all images in advance */
function loadImage(){
    for(let k = 0; k < ItemName.length; k++){
        let item = ItemName[k]; //properties
        let imgArray = []; //keys

        /* For blocks */
        for(let i = 0; i < BlockRowNumber; i++){
            for(let j = 0; j < BlockColumnNumber; j++){
                let index = i * BlockRowNumber + j;
                let img = setPropertyOfImage(item, index);
                imgArray.push(img);
            }
        }

        /* For hold buttom */
        imgArray.push(setPropertyOfImage(item, BlockNumber)); // e.g. index: 36

        /* For current item image */
        imgArray.push(setPropertyOfImage(item, BlockNumber + 1)); // e.g. index: 37

        /* Store */
        imgs[item] = imgArray;
    }
}
```

- 照片的儲存在 imgs，是字典-陣列的二維對應方式，如下圖。
- 透過 appendChild()實現圖片更換。

```
/* Update the information of the block and the image according to "item" */
updateImage(item){
    this.item = item;
    if(this.node.firstChild){
        this.node.removeChild(this.node.firstChild);
    }
    if(this.item != "none"){
        this.node.appendChild(imgs[this.item][this.index]);
    }
}
```

✧ 每個物件有不同的出現機率，該怎麼實現？

- 我的做法是建立物品池 ItemPool，大小 100，有個機率表，表示每 100 個物品會出現多少個該物品。

```
/* 100 in total */
const Possibility = [
    45, 20, 10, 5, 0,
    10, 5, 0, 0,
    0, 3, 2
];
```

```
/* Fill the item pool with 100 images based on the possibility of each item */
function fillItemPool(){
    for(let i = 0; i < ItemName.length; i++){
        for(let j = 0; j < Possibility[i]; j++){
            itemPool.push(ItemName[i]);
        }
    }
}
```

- 全丟進去後，再 shuffle 打散，如此就可以實現不同物品不同體感出現機率。

```
/* Reset the itemPool*/
function resetItemPool(){
    currImgIndex = 0;
    itemPool = FisherYatesShuffle(itemPool);
}
```

- Fisher Yates shuffle 以前沒學過，是從 w3c 上看到，但網站上程式碼似乎有錯。取隨機值 j 的時候，當中的 i 寫錯了，應該是 i + 1。

```
/* Used to shuffle an array in the time complexity O(n) */
function FisherYatesShuffle(arr){
    for(let i = arr.length - 1; i > 0; i--){
        let j = Math.floor(Math.random() * (i + 1)); // 0 <= j <= i
        let temp = arr[i]; // swap arr[i] and arr[j]
        arr[i] = arr[j];
        arr[j] = temp;
    }
    return arr;
}
```

✧ Hold 功能的實作 (其實就是與 Current Item 對調)

- 點擊按鈕後會觸發事件，分成 hold 一開始沒東西和之後有東西。

```
/* Set the event of the hold button */
function setEventOfHoldButton(){
    holdButton.addEventListener("click", function(e){
        e = e || window.event;
        if(holdItem == "none"){
            // Setting hold item
            holdItem = itemPool[currImgIndex];
            updateImageOfHoldItem();

            // Setting current item
            currImgIndex++;
            updateImageOfCurrItem();
        }
        else{
            // Setting hold item
            let temp = holdItem;
            holdItem = itemPool[currImgIndex];
            updateImageOfHoldItem();

            // Setting current item
            itemPool[currImgIndex] = temp;
            updateImageOfCurrItem();
        }
        setLocalStorage(); // store game information to the window.localstorage
    });
}
```

✧ 分數的計算及顯示

■ 物件對應分數表

```
const Points = [
    0, 10, 50, 300, 2000,
    0, 0, 100, 1000,
    3000, 0, 0
];
```

■ 加分的時機有兩個：物件合成升級，點擊分數物件換分數，兩者都會呼叫 addPoints()。

```
const ItemName = [
    "GRASS", "BUSH", "TREE", "HOUSE", "CASTLE",
    "RABBIT", "G8RABBIT", "CARROT", "CARROTS",
    "COIN", "CRYSTAL", "HAMMER"
];
```

```
/* add points */
addPoints(item){
    points += Points[ItemToNumber[item]];
    updatePoints();
}
```

■ UpdatePoints()會更新上方分數資訊

```
function updatePoints(){
    pointDiv.innerText = `${points}`;
}
```

有遇到的困難點，卡了比較久：

- ✧ 如何判斷該物件擺放時需升級？
  - ■ 同物件大於等於三個相鄰即可升級，因此放下物品一瞬間，我們要由上下左右四個方向展開調查，用 BFS 的方式，將相同物品放進佇列裡，直到沒有為止。
  - ■ 同時要記住玩法中允許連鎖升級，也就是升級後剛好可以再升級的狀況，仔細想想最多連鎖一次。我這裡就直接用 while 迴圈反覆檢查，若升級成功會回傳 true 並做下一次的檢查。

```
changeTypeByClick(){
    console.log("-----------------------------");
    console.log(itemPool[currImgIndex], this.index);
    /* 1. Put the item down and check upgrading */
    if(this.item == "none"){
        switch(itemPool[currImgIndex]){ // e.g. "GRASS" (type: string)
            case "GRASS":
            case "BUSH":
            case "TREE":
            case "HOUSE":
                //May combine and upgrade twice successively
                this.item = itemPool[currImgIndex];
                while(this.bfsForCombination(this.item)){
                    this.upgrade();
                }
                this.updateImage(this.item);
                break;
```

  - ■ BFS 我是寫 iterative 版本，因為對 JS 不熟，所以我是用 array 模擬 queue，當 queueIndex<queue.length 表示裡面沒東西了。
  - ■ Dir = [-BlockColumnNumber, BlockColumnNumber, -1, 1]，表示上下左右 4 個方向。
  - ■ 這邊其實就可以看出來，由於 block 是用一維陣列儲存，因此在找尋鄰居時就變得相對複雜。現在看起來似乎也不能說是錯誤決定，畢竟一維也是有一維的好處，但確實是可以想得再周到一些。

```
/* Check if it needs to (upgrade and update neighbors) or (only put down and done) */
bfsForCombination(targetItem){
    let isUpgraded = false;
    let queue = [this.index];
    let queueIndex = 0;
    let sameItemNumber = 0;
    let blocksWaitingForDelete = [];

    let visited = [];
    for(let i = 0; i < BlockNumber; i++){
        visited[i] = false;
    }
    visited[this.index] = true;

    while(queueIndex < queue.length){
        for(let i = 0; i < 4; i++){
            let neighbor = queue[queueIndex] + Dir[i]; //e.g. 26 (type: number)

            // 1. If the index of the block is out of the range => skip
            if(neighbor < 0 || neighbor >= BlockNumber) continue;

            // 2. Blocks in the first column and those in the last column are not neighbors
            if((queue[queueIndex] % BlockColumnNumber == 0) && (neighbor % BlockColumnNumber == 5)) continue;
            if((queue[queueIndex] % BlockColumnNumber == 5) && (neighbor % BlockColumnNumber == 0)) continue;

            // 3. If the block has been visited => skip
            if(visited[neighbor] == true) continue;

            // 4. If the block was "nothing", them it should not be changed in the whole process
            if(blocks[neighbor].item == "none") continue;

            // Otherwise
            if(blocks[neighbor].item == targetItem){
                sameItemNumber++;
                blocksWaitingForDelete.push(neighbor);
                visited[neighbor] = true;
                queue.push(neighbor);
            }
        }
        queueIndex++;
    }
```

- 計算最終相同物品數量，<2 或>=2 有不同的事要做。

```
/* The item clicked by mouse upgrade to be another item */
if(sameItemNumber >= 2){
    isUpgraded = true;
    for(let neighbor of blocksWaitingForDelete){
        blocks[neighbor].updateImage("none");
    }
}

/* The item clicked by mouse just show up as it should be */
else isUpgraded = false;
```

✧ 兔子的各種問題，分成普通兔和反派兔：

✧ 普通兔

- 這邊有兩個問題，一是兔子動畫，二是如何判斷兔子是否要逃跑以及逃跑後胡蘿蔔是否要合成，還有合成位置在哪，這是困擾我最久的地方。

- 先說兔子逃跑的部分。規則是兔子四周若被建築或是分數物件包圍，則會逃跑並留下胡蘿蔔，而若周圍有一塊是空白或是反派兔，則不會逃跑。若要舉個例子，包圍普通兔就像是圍棋吃子一樣，不斷緊氣直到全包圍，可發現相連的兔子是命運共同體，不是一起逃跑，就是都不逃跑，因此如何有效率地找出四散在場上的「兔子群」是一大課題。
- 另外兔子逃跑後留下的胡蘿蔔，也有可能會合成，不論是三隻相連兔子逃跑後所留下的胡蘿蔔合成，還是和場上原有的胡蘿蔔合成，都是需要確認的。
- 我這邊的做法是先分組，逃跑的一組，留下來亂動的一組。

```
/* Process needed for rabbits */
rabbitEscapeAndMove(r){
    let visited = [];
    let needToEscape = [];
    let needToMove = [];
    for(let i = 0; i < BlockNumber; i++) visited[i] = false;
    for(let i of r) this.bfsForRabbit(i, "RABBIT", visited, needToEscape, needToMove);
```

- 找相連兔子同樣是透過 BFS，這邊我覺得寫得好的地方是我把普通兔和反派兔寫在了同一函數裡，然後用 rabbitType 控制，兩種兔子會各自代入不同的參數。
- 一路上會用 rabbitWaiting 記錄相連的普通兔，並沿途用 neighborhood 記錄遇到的鄰居為何。

```
/* Check status of rabbits and G8rabbits */
bfsForRabbit(i, rabbitType, visited, needToEscape, needToMove){
    if(visited[i]) return;
    let queue = [i];
    let queueIndex = 0;
    let rabbitWaiting = [];
    let neighborhood = [];
    while(queueIndex < queue.length){
        let temp = queue[queueIndex++];
        rabbitWaiting.push(temp);
        visited[temp] = true;
        for(let j = 0; j < 4; j++){
            let neighbor = temp + Dir[j];
            if(neighbor < 0 || neighbor >= BlockNumber) continue;
            if((temp % BlockColumnNumber == 0) && (neighbor % BlockColumnNumber == 5)) continue;
            if((temp % BlockColumnNumber == 5) && (neighbor % BlockColumnNumber == 0)) continue;
            if(blocks[neighbor].item == rabbitType && !visited[neighbor]) queue.push(neighbor); // push rabbit into queue if it haven't been visited
            if(blocks[neighbor].item != rabbitType) neighborhood.push(neighbor); // collect neighbors near rabbits
        }
    }
    console.log("neighborhood:", neighborhood);
```

- 若鄰居中有至少一隻反派兔或是空白格就不須逃跑，否則全部逃跑。

- 這邊有個細節是我在群體的最後多 push 一個(-1)，用來表示兔子群的結束。

```
let isEscape = true;

// For rabbits, if there's a space or a G8rabbit nearby, then they will not be scared to escape; otherwise, they will escape.
if(rabbitType == "RABBIT"){
    for(let j of neighborhood){
        if(blocks[j].item == "none" || blocks[j].item == "G8RABBIT"){
            isEscape = false;
        }
    }
}

// Key point: According to the rule, rabbits nearby will either escape or move together.
if(isEscape){
    for(let k of rabbitWaiting) needToEscape.push(k);
    needToEscape.push(-1); // indicate the end of this group
}
else for(let k of rabbitWaiting) needToMove.push(k);
```

- 先討論逃跑的。

```
// escape ("RABBIT" -> "CARROT")
// For those rabbits becoming carrots, we should check if they can combine with each other.
for(let i = 0; i < needToEscape.length; i++){
    let group = [];
    while(i < needToEscape.length && needToEscape[i] != -1) group.push(needToEscape[i++]);

    // Can combine (the rabbit last appearing will upgrade to carrots and others will disappear)
    if(group.length >= 3){···
    }
    // Can't combine
    else{···
    }

    // Maintain rabbitOrder of those escaped rabbits
    for(let g of group) maintainOrder("RABBIT", g, -1);
}
```

- 若該群體要逃跑，假設有 3 隻相連的兔子逃跑了，留下的三根胡蘿蔔會觸發合成升級，但這邊衍伸出一個問題：是哪隻兔子變成的胡蘿蔔要做升級呢？這邊我其實有很多想法，但最後還是堅持必須是最後登場的兔子去做合成升級的動作，因為這樣玩家才能控制胡蘿蔔位置，並盡可能去合成錢幣獲取高分。
- 因此，我需要一個陣列 rabbitOrder 去紀錄兔子進場順序。陣列會儲存兔子所在的格子索引值，先進場的會先被推進去，越後面表示是越晚進場的兔子。
- 同樣反派兔也會有 G8rabbitOrder 紀錄進場順序。

- 我用 maintainOrder()在需要時更新順序資料。其中 oldIdx 和 newIdx 表示兔子從舊位置換到新位置的操作，若 newIdx 是(-1)，表示兔子從 oldIdx 上消失時的操作。

```
/* Maintain the appearance order of rabbits and G8rabbits *
function maintainOrder(item, oldIdx, newIdx){
    if(item == "RABBIT"){
        let pos = rabbitOrder.indexOf(oldIdx);
        if(newIdx == -1) rabbitOrder.splice(pos, 1);
        else if(newIdx != -1) rabbitOrder[pos] = newIdx;
    }
    else if(item == "G8RABBIT"){
        let pos = G8rabbitOrder.indexOf(oldIdx);
        if(newIdx == -1) G8rabbitOrder.splice(pos, 1);
        else if(newIdx != -1) G8rabbitOrder[pos] = newIdx;
    }
}
```

- 如此一來，我就可以知道最後進場的普通兔是哪隻，若有觸發胡蘿蔔合成該是哪個胡蘿蔔要升級。
- 若不逃跑的話，每隻兔子會依序(這裡就沒特別按照 rabbitOrder)找四周空白格，有機率移動一格，反之也有機率原地不動。

```
// move to another block (old position: "RABBIT" -> "none")(new position: "none" -> "RABBIT")
for(let i of needToMove){
    if(i == this.index) continue; // when a rabbit first appears, I don't want it to move
    let neighborhood = [i];
    for(let j = 0; j < 4; j++){
        let neighbor = i + Dir[j];
        if(neighbor < 0 || neighbor >= BlockNumber) continue;
        if((i % BlockColumnNumber == 0) && (neighbor % BlockColumnNumber == 5)) continue; // first column and last column are not neighbors
        if((i % BlockColumnNumber == 5) && (neighbor % BlockColumnNumber == 0)) continue; // first column and last column are not neighbors
        if(blocks[neighbor].item == "none") neighborhood.push(neighbor);
    }
    neighborhood = FisherYatesShuffle(neighborhood);
    blocks[i].updateImage("none");
    blocks[neighborhood[0]].updateImage("RABBIT");
    console.log("rabbit moves from", i ,"to", neighborhood[0]);

    maintainOrder("RABBIT", i, neighborhood[0]);
}
```

- 接著來說反派兔。反派兔是場上沒空格時會一起逃跑，偵測沒空格很簡單，相關合成順序等都和普通兔大同小異。移動的話則是會有機率隨機跳往其他空格，看似也沒有很困難，但和淡入淡出動畫和在一起就是麻煩的開始...。
- 提到動畫，我之所以想做兔子移動的動畫效果是因為原作中有小熊移動的部份，我也發現如果不加點效果，玩家會覺得莫名其妙，為什麼兔子會瞬間移

來移去。由於移動的動作建立在目前寫法上是很難實現的(照片只能固定在格子上)，因此我想到替代方案變是淡入淡出。

- 舉例兔子 A->B，A 點的兔子會逐漸淡出，B 點的兔子會淡入，藉此營造移動的感覺。
- 寫法上我最一開始是用 setInterval()表示每隻兔子的移動，並隨時間調整照片透明度，但我發現了問題就是如果 A->B，B->C，則同一時間 B 可能會淡入又淡出，執行順序未知的狀況下，幾乎保證會出問題。
- 因此我目前的解決方案是以上述為例，A->B 和 B->C，其實就是 A->C，若手動找出 B->B 並剃除，就可以解決問題。

```
// move to another block
else{
    // e.g.
    // g8r = [1, 2, 3, 4], n = [5, 7]
    // candidate = [5, 7, -1, -1, -1, -1], after shuffling => [-1, 7, 5, -1, -1, -1]
    // actual mapping = [[2, 7], [3, 5]]
    //
    // This implementation is pretty redundant though, but it filters out mappings like [2, 2], which will cause problems during fading in/out.
    let indexMapping = {};
    let candidate = n;
    for(let i = 0; i < g8r.length; i++) candidate.push(-1);
    candidate = FisherYatesShuffle(candidate);
    for(let i = 0; i < g8r.length; i++) indexMapping[g8r[i]] = candidate[i];
```

- 若有反派兔分到(-1)，可以變向看成他要移動到任何一隻反派兔準備離開的位置，是我們想跳過的目標。

```
for(let i of g8r){ console.log("g8rabbit moves from", i, "to", indexMapping[i]);
    if(indexMapping[i] == -1) continue;
    let oldPosition = i;
    let newPosition = indexMapping[i];
```

- 否則，那是一次我們可接受的移動，進入動畫部份：
- 以淡出為例：我建立一個 timer，每 0.025 秒會使該格透明度減少 0.1，直到 <=0，就清除 timer 並更換 item 為“none”，同時要記得將透明度再調回 1，原因是我是對該 block 的 firstChild 動手，而且之後該 block 還有可能被用到。

```
// the original block fading out animation
blocks[oldPosition].node.firstChild.style.opacity = 1.0;
let reduceOpacityTimer = setInterval(reduceOpacityToItem, 25, oldPosition);
function reduceOpacityToItem(i){
    let image = blocks[i].node.firstChild;
    let currentOp = getComputedStyle(image).getPropertyValue("opacity");
    if(currentOp <= 0){
        blocks[i].node.firstChild.style.opacity = 1.0;
        blocks[i].updateImage("none");
        localStorage.setItem(`blocks[${i}].item`, blocks[i].item); // need storing additionally since animation happens after setLocalStorage()
        clearInterval(reduceOpacityTimer);
        return;
    }
    currentOp = `${parseFloat(currentOp) - 0.1}`; // remember to convert string to number for calculation
    image.style.opacity = currentOp;
}
```

- 這邊我要坦誠我還沒辦法解決這問題，就是我很想把淡入淡出寫成函數放在外面，但無論如何在清空 timer 時就是無法執行，可能是同時間有多個 timer 在跑，我的理解是 setInterval()宣告時會有紀錄，像是 1 號 timer、2 號 timer 這樣，清空 timer 也是透過該紀錄，但同時我也必須將 timer 宣告在同個定義域內，總之，就是試了很多擺法，也翻了 stackoverflow 的文章，但目前還沒有解決。
- 另外，有發現一個 bug 我也還不知道怎麼處理，就是玩家如果連續按按太快，小於場上兔子動畫所需的 0.25 秒時間，兔子就會發生問題，可能是會消失，或是不怎麼樣，但是 Console 那邊都會跳錯誤訊息出來，這是令我很頭痛的地方。

✧ 如何開啟新的一局？

- 和 hold 相同，都是點擊按鈕後觸發事件，分成六個部份。

```
/* -------------------- About New Game -------------------- */
function setEventOfNewGameButton(){
    newGameButton.addEventListener("click", function(e){
        e = e || window.event;
    1   resetItemPool();
    2   holdItem = "none";
        updateImageOfHoldItem();
        holdButton.innerText = "click me";
    3   updateImageOfCurrItem();
    4   for(let block of blocks) block.resetBlock();
        initializeBlocks();
    5   points = 0;
        updatePoints();
    6   localStorage.clear();
        setLocalStorage();
    });
}
```

- ✧ 如何紀錄遊玩進度，關掉視窗下次打開可以繼續玩，直到按下 New Game 才會換新的一局？
  - ■ 這是我有新增的功能，依靠兩個函式
  - ■ 一是紀錄，二是復原，這邊要注意的是，localStorage 會一直存在瀏覽器直到手動清除，儲存格式是字串，因此若要存陣列就要透過 JSON.stringify()儲存及 JSON.parse()復原
  - ■ 紀錄的時間點有：點擊分數物件換分數後、changeTypeByClick()的最後、淡入淡出動畫要清除 timer 的時候(因為通常程式碼都跑完了，timer 才準備要結束，因此要重新記錄一次)、按 Hold button 後、按 New game 後。
  - ■ 復原的時間點有：開啟遊戲時(重新整理時)。

```
/* Keep in mind that it will store as "string" in the localStorage */
function setLocalStorage(){
    localStorage.setItem("holdItem", holdItem);
    localStorage.setItem("itemPool", JSON.stringify(itemPool)); // use JSON.stringify() to store array as a JSON string
    localStorage.setItem("currImgIndex", currImgIndex);
    localStorage.setItem("points", points);
    for(let i = 0; i < BlockNumber; i++){
        localStorage.setItem(`blocks[${i}].item`, blocks[i].item);
    }
}
```

```
function recoverFromLocalStorage(){
    if(localStorage.getItem("holdItem")){
        holdItem = localStorage.getItem("holdItem");
        updateImageOfHoldItem();
    }
    if(localStorage.getItem("itemPool")){
        itemPool = JSON.parse(localStorage.getItem("itemPool")); // recover JSON string to an array by JSON.parse()
    }
    if(localStorage.getItem("currImgIndex")){
        currImgIndex = parseInt(localStorage.getItem("currImgIndex"));
        updateImageOfCurrItem();
    }
    if(localStorage.getItem("points")){
        points = parseInt(localStorage.getItem("points")); // without parseInt(), it will be seen as a string in the follow-up additions
        updatePoints();
    }
    for(let i = 0; i < BlockNumber; i++){
        if(localStorage.getItem(`blocks[${i}].item`)){
            blocks[i].item = localStorage.getItem(`blocks[${i}].item`);
            blocks[i].updateImage(blocks[i].item);
        }
    }
}
```

整個作品最滿意：圖片是用小畫家親手畫的，雖然花很多時間，但我覺得很值得。

最不滿意：美化上還要多學習，尤其是最後幾堂課教的 canvas 啊都沒用到很可惜。扣掉註解那些的應該勉強達到 500 行要求，但程式豐富度也不夠理想。

心得：

這次其實滿晚才開始動工，報告時間又比較早，因此報告時沒能夠拿出太多東西，所幸距離繳交時間還很長，可以慢慢補充。

這次題目選擇是遊戲，我想也會有滿多人做遊戲方面的主題，程式一路寫下來，我覺得最吃重的真的是設計經驗，因為我是第一次獨自從無到有，生出網頁遊戲，加上對 Javascript 仍是不太熟悉，所以有些眉眉角角(台語)，就要邊做邊摸索，我也認為一個好的網頁介面，除了長相吸引人之外，穩定與兼容性也是頗重要的因素，設想各種可能發生狀況，避免玩家因為不明 bug 卡住之類的，或是某些舊瀏覽器無法執行，甚至遊戲框框跑掉變形等等，做遊戲真的很不容易啊！

另外，我也體會到當程式越寫越長、越複雜，要釐清整體脈絡就會變困難，debug 難度也會增加，此時 coding 的紀律就會體現出來，有好好的縮排分段，有好好的取變數名稱，都會讓一切變簡單。而過程中我也花了很長時間再重新檢視程式碼，想辦法把共同的步驟提取出來，分配好每個函數的任務，盡量一個函式只做一件事且通用化，並想辦法讓他變得更簡潔、更好懂，不過這部分我到現在還是覺得很困難，是未來要繼續練習的地方。

總之，很感謝老師的教導，也感謝當初自己有抽到這門課，最一開始是因為爬蟲看不懂網頁原始碼才希望能更深入了解網頁架構，經過整學期的課後，我也發現了更多網頁設計中有趣的地方，希望之後更繼續摸索並不斷進步！