**Notes for Project**

**Similar game platform:** <https://www.teachyourmonster.org/account/play/tm123?player_guid=3881d559-0ccf-4edf-a6a0-82fa8eabee86&practice=false>

<https://www.turtlediary.com/online-games-for-kids.html>

Things to do:

* Alternate colours
* Consider news page.
* Fill background of logo
* Fix navbar borders
* Give games pause and start function.
* Form for contacting me.
* Fix all game bugs(match that number)
* Consider logo for home page

**Project Requirements:**

**Age-Appropriate Design**: Ensure the website's design is tailored for children aged 3 to 8, incorporating visually appealing elements, bright colors, and animated themes to capture their attention.

**Educational Focus**: Prioritize the inclusion of games that enhance fundamental skills like counting, number sequencing, and letter identification, aligning with the educational goals of the target age group.

**Device Compatibility**: Design the website to be responsive and adaptable to various devices, with a specific emphasis on tablet and iPad compatibility, considering the devices commonly used by children.

**Limited Scope**: Focus the project scope on number and letter games to address the specific educational needs of the target age group, avoiding unnecessary complexity and ensuring a more focused and effective learning experience.

**Gamification Techniques**: Implement gamification elements such as points, scoring systems, levels, and rewards to make the learning experience engaging and motivate continuous progress.

**Evaluation Strategies**: Plan and implement evaluation methods, including user testing, feedback surveys, and performance metrics, to assess the effectiveness of the implemented features in achieving educational objectives.

**Technical Feasibility**: Consider the technical aspects, including the programming languages (HTML, CSS, JavaScript) used in similar successful platforms, ensuring the technical feasibility of the project.

***October***

**Gamification elements:**

* **Points and Scoring System**: Users can earn points for completing tasks or solving problems correctly, encouraging them to strive for higher scores and achievements.
* **Levels and Progression**: Games can have different levels of difficulty, and users can progress to higher levels as they master the concepts, providing a sense of achievement and motivation to continue.
* **Rewards and Badges**: Users can earn badges or rewards for reaching specific milestones, completing challenges, or achieving high scores, fostering a sense of accomplishment and recognition.
* **Input Feedback**: Immediate feedback on correct or incorrect answers helps users understand their progress.
* **Immediate Rewards**: Providing immediate rewards, such as positive animations or sounds, for correct answers can create a sense of instant gratification, reinforcing desired behaviour.

**Optional Gamification elements Features:**

* **Competition and Leaderboards**: Users can compete with others or see their own scores on leaderboards, encouraging friendly competition and motivation to improve their performance.
* **Time Constraints**: Some games can include time-based challenges, where users need to solve problems within a specific timeframe, adding an element of urgency and excitement.
* **Achievements and Challenges**: Users can unlock achievements for specific accomplishments or participate in challenges, providing additional goals to strive for during gameplay.
* **Narrative and Storytelling**: Incorporating a storyline or narrative can make the learning experience more immersive and engaging, motivating users to progress to uncover the story's next chapter.

**Reward system:**

* **Points:** Users can earn points or stars for completing levels or tasks correctly.
* **Level Unlocking:** Completing specific tasks or challenges allows users to unlock higher levels or more advanced game stages. This progression system keeps users engaged as they aim to reach new and exciting parts of the game.

**Optional Reward system Features:**

* **Points II:** Accumulating points can unlock new levels or special features within the games.
* **Virtual Currency:** Introduce a virtual currency system where users earn coins or tokens for their achievements. These coins can be used to purchase in-game items or unlock new game content.
* **Badges and Achievements:** Users can earn badges or achievements for mastering certain skills, completing challenges, or reaching specific milestones. Displaying these achievements can provide a sense of accomplishment.

***November***

**Count That Number:**

1. Time Limit: Introduce a time limit within which the user has to arrange the numbers in ascending order. The challenge increases as the user needs to complete the task within a set timeframe.
2. Increasing Number Range: Expand the number range beyond 1 to 10. For example, you can randomly generate numbers from 1 to 20, 1 to 50, or even 1 to 100. Handling a larger set of numbers requires more attention and cognitive effort from the player.
3. Mixed Number Types: Include not only integers but also decimal numbers or negative numbers. This variation in number types can add complexity, especially if the user needs to arrange a mix of integers, decimals, and negatives in ascending order.

**Optional Complexity Features:**

1. Penalties for Mistakes: Implement a penalty system where the user loses points or time for each incorrect click. This adds pressure to accurately arrange the numbers and penalizes mistakes, making the game more challenging.

UML Diagram:

|  |
| --- |
| CountThatNumber |
| -numbersToDisplay: Set<number>  - clickedNumbers: Array<number>  - timeRemaining: int  - timer: Timer  - gameActive: boolean |
| +generateUniqueNumbers()  + displayNumbers()  + checkNumber()  + updateTimer() |

|  |
| --- |
| NumberContainer |
|  |
| +createNumberElement()  + checkNumber()  + updateTimer() |

(Note: Dependency arrow from NumberContainer to CountThatNumber)

CountThatNumber Class:

**Attributes:**

1. numbersToDisplay: Set<number>: A set to store the numbers to be displayed. Each number in the set is of type number.
2. clickedNumbers: Array<number>: An array to store the numbers clicked by the player. Each number in the array is of type number.
3. timeRemaining: int: An integer representing the time remaining for the game.
4. timer: Timer: An instance of a timer class, representing the game timer.
5. gameActive: boolean: A boolean flag indicating whether the game is currently active (true) or not (false).

**Methods:**

1. generateUniqueNumbers(): A method to generate unique numbers and populate the numbersToDisplay set.
2. displayNumbers(): A method to display the numbers in the user interface.
3. checkNumber(): A method to check if the clicked number is correct and update the game state accordingly.
4. updateTimer(): A method to update the game timer and end the game if the time runs out.

NumberContainer Class:

**Methods:**

1. createNumberElement(): A method to create a number element for the user interface.
2. checkNumber(): A method to handle the logic when a number is clicked.
3. updateTimer(): A method to handle the timer updates.

**Guess That Number:**

1. Difficulty Levels: Implement multiple difficulty levels (easy, medium, hard) where the range of hidden numbers varies. For example, in the easy level, the range could be 1-50, while in the hard level, it could be 1-100.
2. Limited Attempts: Allow the player only a certain number of attempts to guess the correct number. If they exceed the limit without guessing correctly, display a game over message.
3. Time Limit: Add a time limit for guessing the number. If the player doesn't guess within the specified time, end the game and display a message.
4. Negative Numbers: Include negative numbers in the range, making the guessing range from -100 to 100, for instance.

**Optional Complexity Features:**

1. Hints: Provide the option for players to ask for a hint, such as whether the correct number is even or odd, or if it is divisible by a specific number.
2. High Score: Keep track of the number of attempts made to guess the number. Display the high score or the lowest number of attempts required to guess the correct number.
3. Randomized Hidden Number: Instead of generating a fixed hidden number, randomize it each time the player plays, providing a new challenge with every attempt.
4. Sequential Guesses: Implement a rule where players have to guess a sequence of numbers in order, not just the final correct number. For example, they might have to guess 3 numbers in a row correctly.
5. Range Adjustments: Allow players to choose their own range of numbers before starting the game, making it more customizable and challenging.

UML Diagram:

|  |
| --- |
| Guess that Number |
| - secretNumber: int  - attemptsLeft: int  - timer: Timer |
| + checkGuess() : void  + startGame() : void  +Congratulations():void  + OutOfAttempts():void  + TimeIsUp() : void |

|  |
| --- |
| Timer |
| -timeleft: int |
| + startTimer() : void |

GuessThatNumber class:

**Attributes:**

1. secretNumber: int (private) - holds the randomly generated number.
2. attemptsLeft: int (private) - keeps track of the remaining attempts.
3. timer: Timer (private) - handles the countdown.

**Methods:**

1. checkGuess(): void - checks the user's guess and updates the game state.
2. startGame(): void - starts the game, initializing the secret number and timer.
3. Congratulations(): void - displays a message for a successful guess.
4. OutOfAttempts(): void - displays a message when the user runs out of attempts.
5. TimeIsUp(): void - displays a message when the time limit is reached.

Timer class:

**Attributes:**

1. timeLeft: int (private) - keeps track of the remaining time.

**Methods:**

1. startTimer(): void - starts the timer countdown.

**Alphabet Quiz:**

**Parental Control page:**

1. Time Limits and Scheduling: Allow parents to set daily or weekly time limits for gameplay. Implement a scheduling feature where parents can specify allowed play times (e.g., after homework or only on weekends).