Javascript 2D Games Development

We are looking for skilled Javascript game developers to develop **three (3)** small 2D games for us. This game will be embedded into another webpage, so the deliverables will be subject to relatively bizzare requirements.

Games 1 and 2: Board games

Game contents

The following elements are needed:

- A path map in the center with a starting point, an ending point, and some questions (order randomized).
- A dice to roll above the map. After clicking, play an animation of dice rolling and display a random side of the dice when the animation stops.
- A cat icon as the character that moves in the map. After one rolls the dice, the cat automatically moves to the corresponding position. Stay for 0.5 seconds while playing a cheerful sound effect. Then pop-up a question window.
- The question window's contents in the first game (first set, so to speak) are:
  - Game 1 has **11** questions, i.e. the game map has 13 stops in total.
  - Question window may show exactly one among the following three types of questions:
    - Display a picture and 4 words (1 correct and 3 wrong words randomly chosen from the question bank; order randomized) underneath, players need to click the words that match the picture. For this type of question, display the picture on the path map.
    - Display a word and two pictures below, students need to click the picture that matches the word. For this type of question, display the word on the path map.
    - Display a picture and two words below (one is the correct word, the other is the wrong form of word), players need to click the correct word. For this type of question, display the picture on the path map.
- The second board game (set 2)'s question window has only one type:
  - The question audio (indicated `Q_word.mp3`) should be played when the cat lands on some location.
  - The hint audio (indicated `H_word.mp3`) should be played if the player makes a mistake.
  - Game 2 has **10** questions, so the map has 1 fewer stops and counts a total of 12 in total.
  - All questions are the same:
    - display in the question box says "Which picture means `${data.questions[i]}`"?
    - two pictures shown below for them to choose
- The player keeps answering questions until the finish point is reached.
  - if the player is 2 spots away from the finish point and rolls a three, then the game should exit directly.
- Feedback at each question:
  - Correct: play SFX and a "well done!" audio.

    - Incorrect: play SFX and let the player retry.

Please see the four (4) attached pictures for examples of the game's supposed look. The two auxiliary buttons ("play" and "help") shown do not need to be implemented.

The game should keep track of the player's performance, by recording the following two variables:

- The total number of attempts made to answer questions (`n_attempts`), and
- The total number of correct answers made (`n_correct`).

Game 3: Listen-and-Click

Game contents:

The following elements are needed:

- A rabbit on the bottom right corner of the screen, clicking the rabbit plays the question word audio.
- Four easter eggs with words and images on the screen.
- Students have to click the rabbit and listen to what the rabbit says then choose (click) the correct egg accordingly.
- Randomize the order of the eggs for each question.
- The player keeps answering questions until the question bank is exhausted.
- Feedback at each question:
    - Correct:
        - play sound effect and show a green chat bubble for 2 seconds, saying: Well done!
        - dim the background and highlight the egg, play the word audio again
        - then move on to the next question
    - Incorrect:
        - play sound effect and show a red chat bubble for 2 seconds, saying: Try again!
        - play the word audio again as hint
        - allow unlimited trials

Please see the three (3) attached pictures for examples of the game's supposed look. The two auxiliary buttons ("play" and "help") shown do not need to be implemented.

What will be provided

The following will be given:

- question banks and sample data, in JSON format
- all necessary SFX files in mp3 or ogg formats

While it cannot be guaranteed, it is possible that we supply you with graphics as well. In case no graphics could be provided, please look for free assets online and include a `.txt` file containing all necessary attributions.

Technical Requirements and Deliverables

While the choice of JS game engine/implementation method is totally up to you (because you know better about the tools than we do!), there are several peculiar requirements regarding the formats of the deliverables.

The product should be a zipped file containing:

- An HTML file for each game
- CSS files for any styling needed in each game
- Exactly one Javascript game script file per game
- Game assets, sorted into their respective folders for each game
- (Depends) Attribution file containing the asset sources and acknowledgements
- (Optional) Third-party Javascript libraries and dependencies

The **HTML** file should:

- contain exactly one `<div>` that holds the canvas/game. The canvas/game should be hidden when the page finishes loading.
- contain a button, upon the clicking of which will reveal the canvas/game and initialize it.
- optionally use CDNs for third-party libraries. This is not a strict requirement, however.

The **Javascript** file should:

- be a monolithic script handling all the game's logics.
- hard-code the game data one way or another. However, please clearly indicate the entry point of the game, such that we can change it later to dynamically obtain JSON game data using AJAX calls. Similarly, please let us know where are the game data located in the script.
- make it easy to find references to the static assets (images and sounds), so that the URLs to them can be changed later (for the server to serve them in a different way).
- provide a public function that serves as the entry point of the game. When called (on a button click, for example), it will setup the visuals and load the game data and assets.
- provide a public function that will be the *exit* point of the game, i.e. when the game is over, a certain function will be called. We will amend it later to return game scores by AJAX calls.
- indicate where the answers are checked and the scores are counted, i.e. please let us know where will the `n_attempt` and `n_correct` values be updated.

Regarding the programming style, we have very few requirements - every good programmer knows to keep his code readable (please put some code comments!) and tab means spaces (we use four).

Responsiveness is only a good-to-have and is not of significant concern here.