Hangman-CLI Game by Steve Hulme

Hangman Game Project

Technologies Used

- node.js – this is a command-line driven, node application.
- npm – the inquirer and chalk npm modules are used.
- javascript –
    - Constructors.
    - Prototypes.
        - The Object.toString() method is overridden with Word.toString() and Letter.toString().
    - ES6 “fat arrow” functions.

Project Structure: Directories and Files

- hangman-cli/
    - index.js (drives the app)
        - requires hangman.js)
    - package.json (npm modules used)
    - README.md (this file)
- hangman-cli/assets/javascript/
    - word.js (Word constructor and prototype methods).
        - requires letter.js.
    - letter.js (Letter constructor and prototype methods).
    - mlb.js (Array of Major League Baseball team names).
    - hangman.js (the game logic).
        - requires word.js; mlb.js; inquirer and chalk

Project Files

- **index.js instantiates a Hangman game object**, utilizing the constructor in hangman.js
- **word.js contains the Word constructor and its prototype methods.** Since it relies on the Letter constructor and methods, it requires letter.js.
    - **The Word object constructor** creates an array of Letters. The array of Letters is an array of Letter objects. We will use the Word object to hold our hangman game’s riddle.
    - **guessLetter()** looks at the Letter objects in the Word object to see whether the input "char" matches one or more of them.
        - If there is a match, return true.

    - **solved()** is a method that solves the riddle by returning a string in which all letters are visible.
    - **toString()** overrides the Object toString method. It looks at each Letter in the Word object, so that when we output the Word object's letters, we get a string in which each char is separated by
      a blank.
    - **allVisible()** is a method that returns true if every letter is visible. We'll use this method to determine whether the riddle has been solved.
- **letter.js contains the Letter constructor and its prototype methods**.
    - **The Letter constructor builds a Letter object from the input character.** This Letter object has two properties.
        - this.char is the character itself. this.visible is true if the character can be displayed.
        - this.visible will be set to true when a letter in the riddle is guessed by the user. In addition, spaces and punctuation characters are always visible.
    - **toString()** overrides the inherited Object toString() method. Our toString() looks first at this.visible property.
        - If this.visible is true, it returns the underlying character value. If this.visible is false, it returns an underscore character.
        - This method is used to build output displays of the current state of the riddle.
    - **isVisible()** returns true if the Letter object is currently displayable (e.g., if the underlying character has been guessed). Otherwise, it returns the "_" character because the underlying
      character has not been guessed.
    - **guess()** is the method that actually determines whether the input guessed character matches the current this.char.
- **hangman.js contains the Hangman constructor**.
    - This constructor, when instantiated, contains the logic for one game session, which will include one or more iterations of the Hangman game. It utilizes the Word constructor from word.js.
    - In each "Hangman" game the user tries to guess the letters of the Major League Baseball team name that is hidden in the teamName riddle.
    - When it begins, the game picks a team name at random,
    - conceals all its alphabetic letters, and presents it to the user. The user will guess letters in the team name. The user is allowed 10 misses. If the user makes a correct guess, the letter(s) that match the guess are revealed.
    - If the user guesses all the letters in the teamName riddle with less than 10 misses, the user has solved the riddle and wins the game. The user can then continue playing the current game session or quit.
    - Racking up 10 misses without solving the riddle is a loss. The user can then continue

    playing the current game session or quit.
    - As noted, the game tracks wins and losses within the current game session. This is an additional feature beyond the project's requirements.

Pseudo Code for the Hangman CLI game

Initialization:

- initGame is the starting point for a new game session (a new Hangman object) . This function sets the number of wins and
  number of losses to zero and then calls the nextTeam function.
- nextTeam picks a random team name, turns it into a Word object (using Word constructor), then displays the team name as a string.
    - Alphabetic letters (letters to be guessed) are displayed as "_". Spaces and punctuation characters are always visible. nextTeam then calls guessTeam.

Guessing:

- guessTeam is the function that contains the guessing logic. It calls getKeyStroke to get the user's next guess.
- After the guess (whether correct or incorrect). the game will present the current state of the hidden team name.
    - Letters in the teamName riddle whose "visible" property is "true" (set by getKeyStroke) will be displayed.
- Next, the game evaluates the state of play:
    - If number of guesses remaining is 0, game is over and the user has lost.
        - The number of losses is incremented.
        - Current wins/losses are displayed.
        - The user is asked whether the game should continue.
        - If user wants to continue, the game instance makes a recursive call to its nextTeam logic to get a new team name to guess.
        - If user does not want to continue, the game calls the exit logic in endGame.
    - Otherwise the number of guesses remaining is not 0 and so the
      user has not lost.
        - If all the alphabetic chars in the teamName riddle are visible, the user has won. We figure this out by using the Word.allVisible() method.
            - The game increments the number of wins, displays current wins and losses, and asks the user whether the game should continue.
                - If user wants to continue, the game makes a recursive call to its nextTeam logic to get a new team name to guess.
                - Otherwise, the game calls the exit logic in endGame.

        - Otherwise, the game knows that there are letters remaining to be guessed, so it makes a recursive call to the current game's guessTeam function.

getKeyStroke

- getKeyStroke gets the user's next guess (using inquirer).
- It then checks to see whether the letter guessed appears one or more times in the teamName riddle (using the Word.guessLetter() method).
    - If the user has guessed a letter that appears in the team name, the "visible" property of the corresponding letters is set to "true" using a Word method.
    - Otherwise, the guess is incorrect and so the number of guesses is decremented by 1.

endGame

- The final tally of wins and losses is presented to the user.
- The game thanks the user for playing.
- process.exit is called, with a “success” status code of 0.