Monster Portal: A Game That Is Totally Dissimilar to Hangman.

A picture containing text

Description automatically generated  A picture containing outdoor, rock

Description automatically generated

A picture containing text, outdoor

Description automatically generated A picture containing outdoor, tree, cave

Description automatically generated

A picture containing outdoor

Description automatically generated A picture containing nature, ravine

Description automatically generated

**Table of Contents**

[*Wireframe 3*](#_heading=h.gjdgxs)

[*Layout Notes: 3*](#_heading=h.30j0zll)

[*Constants 3*](#_heading=h.1fob9te)

[*Classes 4*](#_heading=h.3znysh7)

[class letter {} 4](#_heading=h.2et92p0)

[class word {} 4](#_heading=h.tyjcwt)

[class secretPassword {} 4](#_heading=h.3dy6vkm)

[class keytopObject {} 4](#_heading=h.1t3h5sf)

[class keyboardObjectArray {} 4](#_heading=h.4d34og8)

[class activeSecretPassword {} 4](#_heading=h.2s8eyo1)

[class demoScript {} 4](#_heading=h.17dp8vu)

[*Unique IDs 4*](#_heading=h.3rdcrjn)

[*Pseudocode: 4*](#_heading=h.26in1rg)

[***Fridge Stuff 6***](#_heading=h.lnxbz9)

[*Sounds: 6*](#_heading=h.35nkun2)

[*Local.Storage usage 6*](#_heading=h.1ksv4uv)

[*Demo Mode: 6*](#_heading=h.44sinio)

[*Misc. 6*](#_heading=h.2jxsxqh)

[*SecretPasswords 7*](#_heading=h.z337ya)

[*Fridge List 8*](#_heading=h.3j2qqm3)

## Wireframe

Diagram

Description automatically generated

## Layout Notes:

* Monster: 2\*4 grid, top row is span 2.
* Letters: 13\*2 grid, 20% radius corners.
* Click on letters turns them ‘green’ or ‘red’

## Variables, Constants, & Game States

* badGuesses
* correctGuesses
* totalGuessesMade (7 – badGuesses) + correctGuesses;
* gameWins
* gameLosses
* guessRemaining
* nextMonsterTile = “mp1.jpg”; /\* default \*/
* howToPlayURL = “html/howtoplay.html”; /\* Help doc must include link back to game \*/
* playTotallyOriginalThemeSong = false; // default, bool
* gameTitleMessage = “Welcome to the Monster Portal!” /\* Game messages such as correct letters, Game wins & losses, and impending seismic events are displayed here. \*/
* gameTitle = “MonsterPortal™”; // game title. Duh.

## Classes

### class letter {}

### class word {}

### class secretPassword {}

### class keytopObject {}

### class keyboardObjectArray {}

### class activeSecretPassword {}

### class demoScript {}

## Unique IDs

* Each keytop has a specific item id, as in id=”keytop-A”, etc. Not sure if this is a good idea, but it allows me to identify them immediately, even if I later decide to implement a QWERTY style keyboard layout to render an old “Cold War-style keyboard into the game.

## Pseudocode:

* MonsterPortal Game link is invoked in the browser, index.html page is rendered.
* initializeGame() is called: All initial variables, constants, static arrays, objects, complex structures, etc. are created and initialized as defined elsewhere in this docuyment.
* renderGame() is called: The user is shown the Monster Portal game page in initial state. Within this function:
  + showScoreboard() is called: There is empty scoreboard on the left, listing correct and incorrect guess, and games won and lost. Initially, all zero except BadGuessRemaining, on each subsequent call showScoreboard() display the most recent values for the various states and values they represent.
  + showGameMessage(): The game’s Title and Initial Greeting message displays along the top of the screen. On each subsequent call, this will reflect any new messages that have been generated.
  + showMonsterPortal(): An empty framework broken out into 7 segments is displayed in the middle of the screen. On subsequent calls, showMonstorPortal will display a single MonsterTile ( image tiles numbered mp1.jp through mp7.jpg)
  + showMenu(): the simple game menu is displayed on the upper right of the screen, showing 4 menu items, rendered vertically:
    - How to Play
    - Start Game
    - Reset Game
    - Quit
    - It pretty much looks the same each time.No state changes require it to be re-renderd except click events, which are handled by attaching event handlers.
    - [Optional] There may also be a media bar that allows the player to turn on the TOTALLY ORIGINAL THEME SONG, which, (and I can’t express this enough), is ***totally*** and ***completely* original**, and created ***specifically*** and **solely** for the purpose of playing as dramatic background music for this game. The Monster Portal theme song is called ~~“sounds/mission-impossible-theme-song.mp3”~~ “sounds/monsterportal-themesong.mp3.”.
  + showPasswordField(): Beneath the empty MonsterPortal frame, there are a number of line segments that are the width of an ‘xx-large’ font sized letter. These represent the letters that you’re trying to guess for the “Secret Password” you are trying to “hack”, in order to prevent the portal from opening completely.
  + showKeyboard(): This renders the 13 \*2 grid of keytops that the user will desperately click in their desperate attempt to “hack” the password and prevent the portal from fully opening.
* When the user clicks the “Start Game button, the renderGame() function is called.
* When the user demonstrates their elite hacking skills by guessing a letter, the isGuessCorrect() is called. At this point, one of two things will happen:
  + If they make a correct guess, the keytop turns green, a happy bell sound plays, and the correct letter is displayed in each appropriate location of the Secret Password field.
    - If there are no more blank letter spots remaining, the user wins the game. A celebratory song is played, and dancing ensues.
  + If they make an incorrect guess, the keytop turns red, the guessesRemaing is decremented and compared to zero, and a harsh, judgemental tone is played.
    - If the guessesRemaining reaches zero, the player has lost the game. ~~They are unceremoniously dumped on the floor and beaten.~~ A sad noise is played to express the entire planet’s disappointment at facing a horrible death by being eaten by Portal Monsters. Inexplicably, GameStop stock rises.
    - Losing letters call these functions: showPiece1() through showPiece7() in the grid, corresponding to the bad guesses index.
* The End. Now let’s write this game.

# Fridge Stuff

## Sounds:

* ding.mp3: Correct guess.
* cowbell.mp3: incorrect guess.
* sosad.mp3: Game loss.
* cheer.mp3: Game Win.
* quit.mp3: **only** if “Quit” is implemented.

## Local.Storage usage

* DemoGameWins initialized to 0, persisted to local.Storage. Toggled with each Demo game.
* gamesWon initialized to 0, and persisted to local.Storage.
* gamesLost initialized to 0, and persisted to local.Storage.

## Demo Mode:

1. A variable DemoGameWin is created and initialized to false.
2. The Demo button uses a demoGame() function to run a series of setTimeout(pressKeytop() actions that post ‘click’ messages to the keytops at a reasonable pace, following predetermined script using a demoScript object, either the demoWinScript or demoLoseScript.
3. Upon first use, the Demo game simulator loses.
4. The DemoModeWin variable is then toggled to ‘true’ and next game Wins, then toggles back to false. Also, if it doesn’t exist, DemoModeWin is written out to localStorage.

## Favicon:



## SecretPasswords

* List of Monster Portal Employee’s Secret Passwords.
* each SecretPassword{} object is stored as the word itself, plus the complete SolutionObject{} for that word, in the database.
* Their IT Department requires them to have a commonly known word, length of between 5 and 9 letters.
* Only upper case alpha characters are allowed in Monster Portal Employee passwords.
* No numbers, special characters, symbols, or even lower case letters were permitted.
* Note: It has often been speculated that this is why Monster Portal security was cracked in 3.7 seconds, using handheld portable programmable calculators.
* The whole list of 20 Employee passwords is stored locally. Clouds didn’t exist at the time the Monster Portal was built, thus their perpetually sunny dispositions. Here is that full list, copied here to keep it safe and secure:
* Translation: SecretPasswordCollection{} will be stored locally at the bottom of the .js file for convenience.
  1. MARSUPIAL = 9 letters.
  2. MOUNTAIN = 8 letters.
  3. MONKEY = 6 letters.
  4. SPATULA = 7 letters.
  5. BICYCLE = 7 letters.
  6. LUNCH = 5 letters.
  7. PIZZA = 5 letters.
  8. Orangutan = 9 letters.
  9. Shovel = 6 letters.
  10. Syllable = 8 letters.
  11. Canine = 6 letters.
  12. Table = 5 letters.
  13. Conflict = 8 letters.
  14. Parade = 6 letters.
  15. Wizard = 6 letters.
  16. Pandemic = 8 letters.
  17. Theater = 7 letters.
  18. Desert = 6 letters.
  19. Forest = 6 letters.
  20. Surfboard = 9 letters.

## Misc. Random Stuff

* revealSlots(SecretPassword.length); OR show slots from beginning to SecretPassword.length, and don’t worry about centering.
* slot1.visible = true, .visible = false;, etc
* Attach password letters to cells above each visible slot.
* Use reveal.GuessedLetter(); upon each successful guess.
* Use array.includes(thisElement) to determine if const currentGuess = “default”; is correct.
* “Start Game”: Spawns a new tab with window.Open(), so that when Quit is pressed, it will play the quit() sound and close tab.
* <https://stackoverflow.com/questions/11922383/how-can-i-access-and-process-nested-objects-arrays-or-json>
* M.I. Theme Music? “Cant afford the licensing fees…”
* ~~Comic Sans font.~~ Neucha, because Comic Sans is $50 for each font type.
* Store the length of the password as a property of the SecretPassword object.
* Add a Difficulty rating as a property to the SecretPassword object class.

## Fridge List

