CS385 Mobile Application Development

Hangman Dude - Project Report

By Team Cayrel: Michelle Campion, Nelya Nyegutsa and Oleksandr Chepyha

Chapter 1 - Why did we develop this app?

On forming our group, we held a brainstorming session during which, we all put forward different project ideas and discussed the pro's and con's of each suggestion. We all agreed on the most interesting and exciting option, and decided to create our own version of the well-known game "Hangman".

The reasons we decided on our app "Hangman Dude" are as follows:

- We thought working on a game would be a fun way to implement some of the tools we have learned over the semester in React.
- We thought Hangman was a good game to work on as it is straightforward to play and easy to understand.
- This game could appeal to a wide range of people, adults and children alike. For adults we felt that there might be a nostalgic feeling towards the game and that we could create some renewed excitement about it too, by adding our own unique, creative images to it.
- We thought we could make it look visually attractive, with bright colors and that we could make our images cartoonish to appeal to children.
- We searched for an API that we could use to generate words for the game, but we found an array of approximately 3,000 words that we saved as a javascript file, and so felt we had a good database there that we could use to run the game.

We used a website *Namelix.com* to help us with the task of choosing an app name. We typed in the keyword "Hangman", it generated lots of names from which we selected a few favourites and finally settled on "Hangman Dude".

During this brainstorming session, we also tried to draw a basic wireframe design of the different components we could create for our app, and also to list the functionalities the app would have.

Once we had discussed this is detail, we all felt that we had an understanding of how we wanted our app to work and a very general idea how we wanted it to look. The development in the visual design of our app from where it started to where it is now, has been ongoing, based on lots of testing, user feedback and many group meetings.

Chapter 2 - How did we design this app?

As a group, we decided to have weekly meetings and use online resources, namely *Slack.com* and *Trello.com*, where we could assign tasks, share work and ideas, and have group discussions regarding any issues or development suggestions.

In this way, we were all aware of each other's progress and could help each other with any challenges.

During our group meetings, we discussed in great detail how we wanted this app to look. As mentioned above,

- we knew we wanted it to appeal to people of all ages
- we wanted to use bright, appealing colors
- we wanted to have a user-friendly interface and functionality, with all the components of the game very clear and easy to see
- we continued work on our wireframe model of the app functionality to plan how all the components would be displayed.

When discussing the visual concept, we decided we wanted to give our game a school-based theme, as it is an educational game that most people first learn to play in school. The images we created to display throughout the game were all designed on this basis, using an image of a notebook to display the app name, and for all other images we use a crumpled paper effect with pen writing/drawings.

Our graphics were created in Adobe Photoshop with the help of standard inbuilt filters and special effect techniques.

The User Experience:

We wanted to provide game players with a different experience than the traditional Hangman game. Traditionally, when playing, the gallows are drawn first when the player guesses wrong, followed by a stick man's head, and body etc.

We decided it would be nicer to start by drawing our cartoon stick man first, and have his expression change gradually as the player's guesses remaining decrease, and then only to include the gallows in our game over image, if the game is lost.

We decided to display all the letters of the alphabet as buttons in a table, as we thought this gave players the most user-friendly experience. In the early stages of the app, these buttons were displayed in a row underneath the game image, but this resulted in having to scroll down to see the full page. To fix this, we redesigned the layout so that the game image is on one side of the screen and the alphabet buttons, guesses remaining etc are displayed on the other side of the screen. We did this using Bootstrap grid layout and this resulted in a much more visually pleasing display. We also updated our background in our App.css file to include a gradiant color.

Chapter 3 - How did we implement the functionality in this app?

We decided to display our game using conditional rendering in our App Component. This means that at any stage of the game, one of four possible screens will be displayed. For the purpose of this report we will name these display options as follows:

- Start screen
- In-game display
- Winner screen
- Game over screen

Start screen

Upon opening the app, the start screen is displayed. On this screen, we have our game logo, a drop-down list for the player to select their difficulty level and a play button to start the game. The condition for displaying this start screen is that the play button has not yet been clicked.

This play button is disabled until the player has selected their desired difficulty level. The drop-down list provides a description of the possible word lengths associated with each level. Once the difficulty level has been selected, we use a filter function to filter our words array based on the word lengths as outlined. The new array of words output from the filter function is then assigned to a state variable *words* and will be used to generate a word for the game.

The play button can now be clicked. Once clicked, the app will then render the in-game display. However this play button does handle a few other tasks behind the scenes. It calls *three* other functions (as follows) to assign values to state variables before starting the game:

1. *generateRandomWords* - which selects a random word to use for the game. It takes in our state variable *words* as a parameter, generates a random index and returns the word at that random index.
2. *wordToArrayOfLetters* - which takes this random word (above) and creates a

character array representation of this word. We've done this so we can easily compare each guess the player makes with each slot of this array (and so we can easily update the next array mentioned below in function 3 which will hold correct user guesses).

3. *startingDashes* - this function basically creates a copy of the character array above but instead of each array slot holding a letter from the random word, it instead holds an underscore line (_) to represent a letter that has not been guessed yet. This array will be rendered in our in-game display and illustrates to the player the total number of letters in the word, and how many letters there still are to find whilst playing. It is a state variable and will be updated as the player guesses letters correctly.

In-game display

As mentioned above, the next screen that is rendered after clicking the play button is our in-game display. This screen is rendered while **both** the following conditions remain true:

1. guessesRemaining > 0

   (*when this condition is false: the game over screen is displayed)*

2. there are still more letters to guess (ie. the array displayed to player still contains underscore characters)

   (when false: the word has been guessed => winner screen is displayed)

On this screen, our render function displays our 'ImageDisplay' component. This component holds all of our in-game images and displays them in sequence based on the number of guesses remaining. We have done this by putting the image names in an array so that the index for each image represents the number of guesses remaining when that image is displayed.

For example, when guesses remaining is 11, the head image, which is held at index 11 of the image array, is displayed.

Another component we display in this in-game screen is our 'Letters' component (alphabet buttons as mentioned previously). These alphabet buttons use a very important function in the game: *checkWordArrayForGuess.* This function:

- identifies the letter the player has chosen by it's 'letter id' (note: the letters are stored as an array of objects in the *Alphabet.js* file)
- disables the letter chosen so that it cannot be chosen again (boolean variables also stored in an array in the *Alphabet.js* file)

- it checks the *randomWordArray* state variable for any slots that have that letter and if so,
- updates the state variable *dashLine* array with this letter in the corresponding space(s)
- if the letter chosen is not in the word, the guesses remaining are reduced by 1.

Along with these components, this screen also displays the state variable *dashLine* (and it's updated values progressively) and the number of guesses remaining for the player (which is another state variable *guessesRemaining*). We also decided to add a 'Quit Game' button here so that the player does not have to go through the whole game to get back to the start screen. This button uses a modal to ask the player if they are sure they want to quit.

Winner screen

If the player manages to guess all of the letters in the word correctly, our conditional render function will then display our winner screen. This screen displays our 'Winner' component which holds the winner image and also displays the word of the game.

This screen also gives players the option to play again (another button).

Game over screen

If the guesses remaining reach zero, unfortunately it's game over and finally our gory hangman image is displayed by calling our 'GameOver' component. This screen also tells the player what the game word was and gives them the option to play again (button as on winner screen).

The play again buttons on the winner screen and the game over screen, and also the quit game button on the in-game display screen all use the same function: *resetGame()*.

This function just sets all state variables back to their original values.

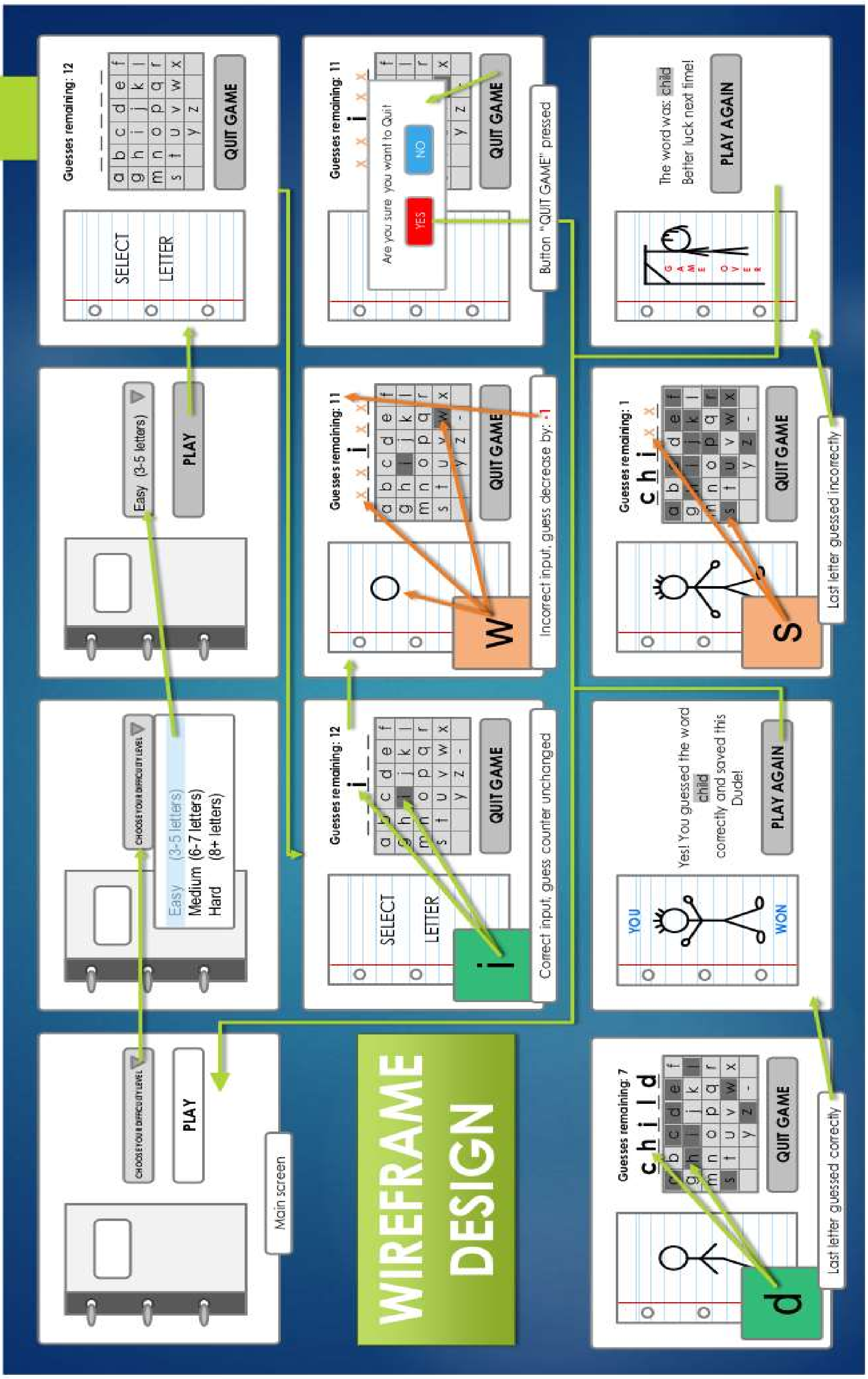
Chapter 4 - Overall Evaluation

Overall we are happy with the progression of our app. We are pleased that the game functions correctly and looks a lot better visually than where it started. The nice thing about our app being a game is that it was relatively easy to test, as this just involved a lot of playing. We played it ourselves a lot throughout it's progression, thinking about it's layout and what we could improve, and also had friends and family play it too and provide their feedback which was all very useful in coming up with the design.

Things like adding the 'Quit game' button and choosing the background color etc were decisions based on this testing and user feedback.

If we had another 4-6 weeks, there's some extra functionality we would like to add to our application. We thought about giving users of the app the option to log in so that they could keep track of their wins and losses, and maybe even to have a leaderboard accessible to view by all players. Another idea we thought might be nice to add would be to give players the option to choose a theme and generate the random word from a collection of words relating to that theme, eg. sports, animals etc.

For the time we had however, we are very happy with how our Hangman Dude game has turned out, and have enjoyed the process of creating it. We hope you enjoy Hangman Dude too.



WIREFRAME DESIGN
Main screen
CHOOSE YOUR DIFFICULTY LEVEL
PLAY
Easy (3-5 letters)
Medium (6-7 letters)
Hard (8+ letters)
Guesses remaining: 12
SELECT LETTER
QUIT GAME
Correct input, guess counter unchanged
Guesses remaining: 11
Incorrect input, guess decrease by: -1
Are you sure you want to Quit
YES
NO
Button "QUIT GAME" pressed
The word was: child
Better luck next time!
PLAY AGAIN
GAME OVER
Guesses remaining: 1
Last letter guessed incorrectly
YOU WON
Yes! You guessed the word child correctly and saved this Dude!
PLAY AGAIN
Guesses remaining: 7
Last letter guessed correctly