[Hack] - official Documentation

Grupp 16

June 6, 2025

Contents

1 Introduction

The game built is a implementation of the fallout 3 hacking minigame. The games is a text game where you try to guess the correct word from different words in a mega array map. The game starts the user with 4 health that are indicated by 4 blinking lights. When the user guess the right word a win screen begins to flash. If the user losses all health on wrong guess then the lose screen begins to flash.

2 Gameplay

2.1 Game rules

RULE 1 : Each guess has to start with a > sign before the word

RULE 1 : The putty window must be 24 x 69 for the game screen to appear correctly.

RULE 3 : Guess only one word at a time

2.2 Controlls

Controll	**Function**
[Keyboard]	[strings/words guessed by user]

Table 1: controlls

2.3 Goal of the game

The goal of the game is to guess the correct word without running out of health.

3 System

3.1 Blockdiagram

The following diagram shows the system architecture and the flow of communication:

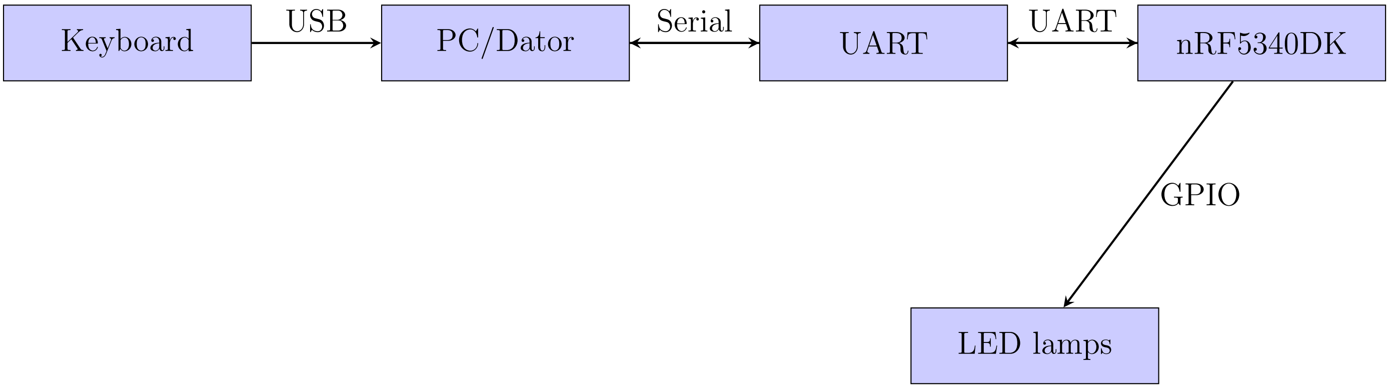

Figure 1: System architecture and flow of communication

4 Code implementation

4.1 Main loop

The following box diagram describes the main gameplay loop:

.

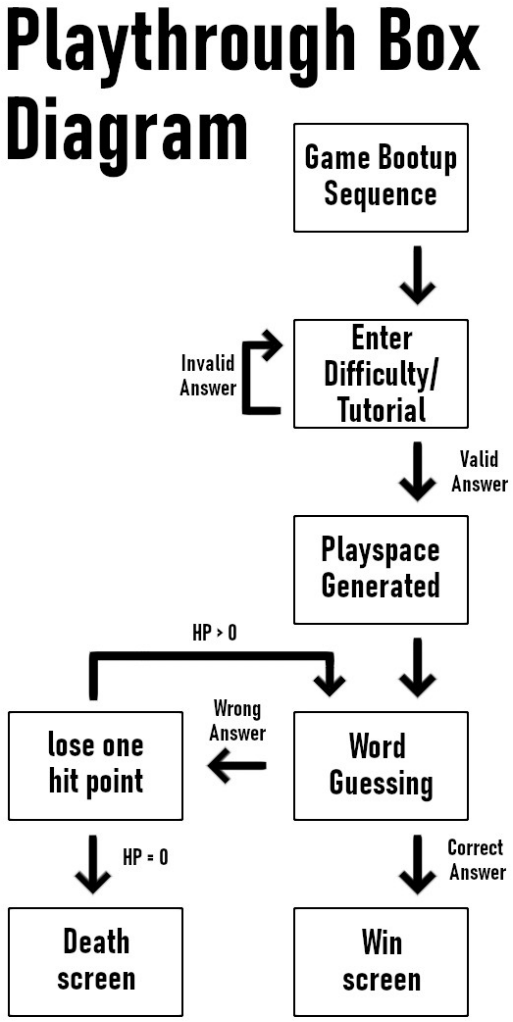

Figure 2: Flowchart depicting the playthrough flow from the users perspective. With each row of boxes being a distinct phase of code execution.

5 Game logistics

5.1 Game Bootup Sequence

STEP 1: Global uarte instance is initialized.

STEP 2: Rx is armed so user input can be listened for and recived.

STEP 3: Health is initialized.

STEP 4: Dev startup splash screen is played.

STEP 5: A char array containing the entire difficulty select screen is generated and then gradually filled, milestone changes result in display update.

STEP 6: A blank char array (called gameScreen) is allocated memory, used later for the playspace.

5.2 Enter Difficulty/Tutorial

STEP 1: When user enters their answer, Rx saves it in a DMA buffer and iteratively moves the char from the DMA buffer to a char array ready to be compared with a valid word, then re-arms rx and then try to resets the peripheral error registry.

STEP 2: if no valid difficulty was selected then re-arm rx and try again.

STEP 3: If user enters valid difficulty then game state changes and the Playspace is generated.

5.3 Playspace Generated

STEP 1: A list (called Deck) that will contain every word in play is declared.

STEP 2: Every word contained in char array lexicon in easyLexicon is inserted as a node containing an ID and a word into Deck. The amount of words in lexicon functions as a seed determining how word removal in shuffleWords, character distribution in genJunk, and word seeding to generated pillars play out.

STEP 3: shuffleWords selects which word ID the user has to guess from the Deck. After which it removes nodes from the Deck until only 20 words remain.

STEP 4: Deck is sorted with a bubble algorithm for ease of navigation.

STEP 5: ID is used to fetch the word that the user has to guess.

5.3.1 Playspace GUI

STEP 6.1: Two pillars containing a selection of characters are generated and inscribed to gameScreen.

STEP 6.2: All the words contained in Deck are distributed into the previously generated pillars.

STEP 6.3: The Pre defined char arrays are inscribed to gameScreen.

5.4 Word Guessing

STEP 1: The health will constantly get checked and while it checks the health it also controls the blinking lights.

STEP 2: When user enters guess the program goes through the same logic as in ”STEP 1” in (Enter Difficulty/Tutorial) where it uses a DMA buffer and comparisons.

STEP 3: If the guess is wrong then the program goes to Lose one hit point.

STEP 4: If the guess is right then the program goes to Win Screen.

5.5 Lose one hit point

STEP 1: the user loses 1 health or if health is less than or equal to 0 then they will go to the death screen.

5.6 Win Screen

STEP 1: The screen flashes with the text ”You’re Winner!

5.7 Death screen

STEP 1: The screen flashes with the text ”You are dead lel”