Consonant Keyboard: Midpoint Report

Mark Lavrentyev Jed Fox Max Heller Kris Diallo

November 9, 2022

1 Project overview

The consonant keyboard project aims to create a functional keyboard allowing users to type as they do on a normal keyboard, except the vowel keys will be omitted. The user will only need to type the consonants of the words they intend to type, and the keyboard will attempt to replicate the original word the user intended once the space bar is pressed. While largely a novelty item, this could be useful in speeding up the typing speed of users who get enough practice with it.

2 Requirements

R1. The device shall be a normal (alphabet-only + space bar) QWERTY-layout keyboard, with vowel keys ('a', 'e', 'i', 'o', 'u', and 'y') omitted.

R2. The device shall support being attached via a USB cable to any computer that supports ordinary USB keyboards.

R3. The device shall read all non-space keypresses and echo them to the computer when the key is pressed down.

R4. When a key is held down, it should only be interpreted as a single keypress.

R5. When the space bar is pressed, the device shall attempt to fill in the missing vowels by connecting to GPT-3. When the response is received, the characters already entered should be removed by simulating repeated presses of the backspace key, and then the completed word should be entered in via simulated keypresses.

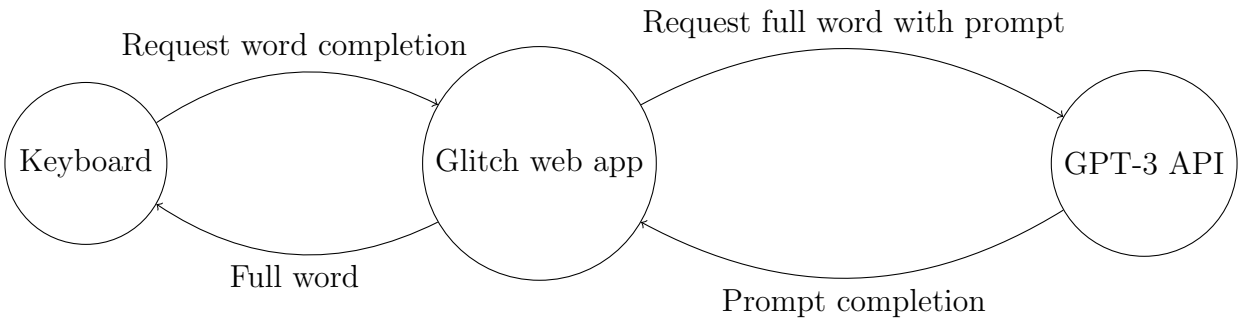
R6. If the time to fill in the missing vowels exceeds 10 seconds, the device shall simulate typing out an error message for and subsequently should reset itself.

R7. If the space bar is pressed while the user has no letters entered in the buffer, there will be no effect.

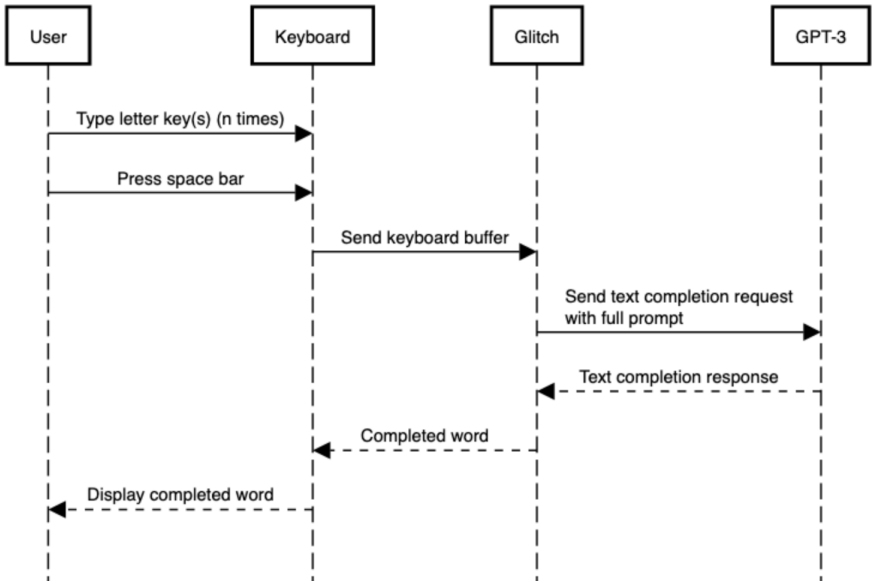
R8. If the user presses multiple letter keys at the same time, the device will ignore letter keypresses until all letter keys have been released.

R9. If the user presses the space bar and a letter key at the same time, each letter shall be added to the buffer and the resulting buffer shall then be sent for vowel insertion.

3 Architecture



4 Scenario



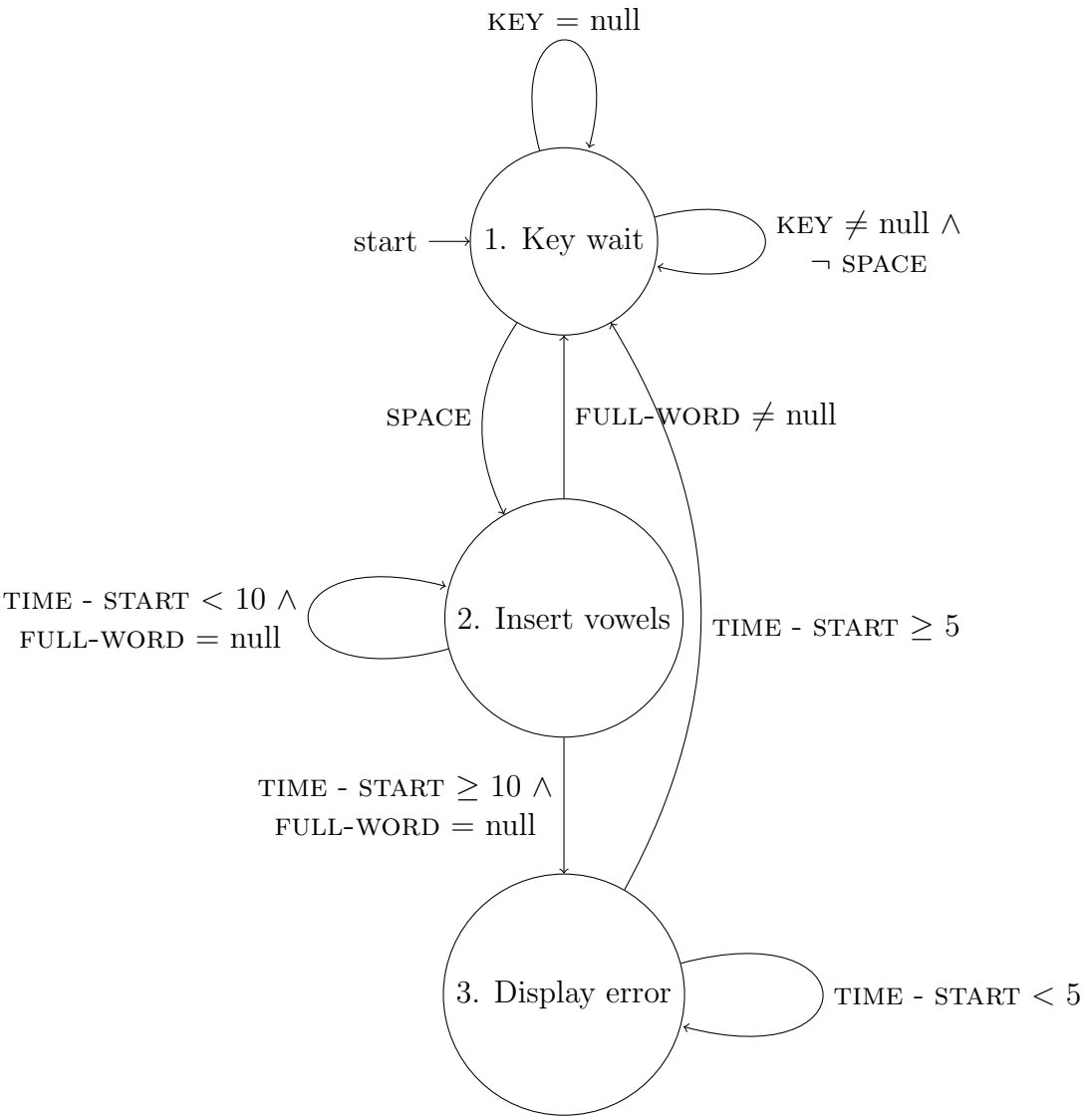
5 Finite state machine model

Our finite state machine will be hybrid time- and event-driven. It needs as input the current time (TIME), the currently pressed letter key (KEY, equal to "null" if nothing is pressed), and whether space is pressed (SPACE). In state 1, it will be run whenever the KEY or SPACE inputs are changed. In state 2, it will be run every 10ms.

It has the following variables:

- BUF: the current letters the user has typed in, but not yet sent for vowel insertion.
- START: time since the state started (for states 2 and 3).
- FULL-WORD: the full word, with vowels inserted.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
1-1a	X								
1-1b	X		X	X			X	X	X
1-2				X	X			X	X
2-1					X				
2-2						X			
2-3						X			
3-1						X			
3-3						X			



Transition	Guard	Explanation	Actions
1-1a	$\text{len}(\textsc{keys}) = 0$	No keys pressed.	-
1-1b	$\text{len}(\textsc{keys}) > 0 \land \textsc{space} \notin \textsc{keys}$	Some letter keys were pressed.	$\textsc{buf} \mathrel{+}= \textsc{keys}$
1-2	$\textsc{space} \in \textsc{keys}$	The space bar was pressed.	$\textsc{buf} \mathrel{+}= \textsc{keys}$ - " "; $\text{recreate}_{w}ord(\textsc{buf})$; $\textsc{buf} = ""$; $\textsc{start} = \text{current}_{t}ime()$;
2-1	$\textsc{full-word} \neq ""$	The recreated word was generated.	$\text{display}_{f}ull_{w}ord(\textsc{full-word})$;
2-2	$\textsc{time} - \textsc{start} < 10 \land \textsc{full-word} = \textsc{null}$	Waiting for recreated word.	$\textsc{full-word} = \text{get}_{r}ecreated_{w}ord()$;
2-3	$\textsc{time} - \textsc{start} \geq 10 \land \textsc{full-word} = \textsc{null}$	Request to recreate word timed out.	$\textsc{start} = \text{current}_{t}ime()$;
3-1	$\textsc{time} - \textsc{start} \geq 5$	Reset the system.	$\textsc{buf} = ""$; $\textsc{full-word} = ""$;
3-3	time - start < 5	Displaying error message.	$\text{display}_{e}rror()$;

6 Keyboard handler

The state machine above does not contain the logic for decoding keypresses. We will be arranging our keys into a virtual 5×4 grid (plus an additional input for space). Each column, along with space, will be connected to an interrupt handler. Whenever one of these interrupts is triggered, we will update KEYS and SPACE using the following algorithm:

- If the space key is pressed, set SPACE to TRUE, FALSE otherwise.
- Otherwise, if exactly one column and exactly one row is active, set KEY to the character at that row and column in the key map.
- Otherwise, set KEY to null.

7 Test plan

There are essentially two main subsystems that are interacting in this project: the keyboard (and reading user keystrokes into a buffer), and the GPT-3-based word recreater. To unit test the keyboard, we will test the following scenarios:

- Test that no action occurs when no keys are pressed (via passing empty KEYS input).
- Test individual key pressed (passed into FSM KEYS) for each key and verify that the right letter is added to the buffer.
- Test that multiple simultaneous key presses (without space bar) add no letters to the buffer. Verify that this works for any length of KEYS, from 2 to 20.
- Test that the space bar advances the FSM to state 2.
- Test that pressing the space bar along with a letter adds that letter to the buffer and then moves the FSM to state 2.
- Test that receiving a recreated word types it out and returns the FSM back to state 1.
- Test that if the GPT-3 request times out, an error is displayed (advances to state 3).
- Test that before 5 seconds, the error continues to display (transition 3-3). After 5 seconds, the system is reset (3-1).

For the GPT-3 subsytem, we will implement the following unit tests:

- Test that the correct prompt is built given a consonant-only word.
- Test that a GPT-3 API request is sent with the built prompt.
- Test that the completion word is correctly extracted from the API response.

For end-to-end system tests, we will test using a user typing normally. We expect that the user will be able to type with consonants only, and have the word completed when pressing the space bar.

8 Reflection & updates

We met our goals set out for the midpoint milestone that we set out in the original project proposal. In particular, we currently have a working proof-of-concept circuit with a few keys (not the whole keyboard layout yet), allowing us to validate that our proposed circuit for connecting 20+ keys to the limited GPIO ports on the Arduino will be possible. In addition, we have a working connection to the GPT-3 API via WiFi, allowing the Arduino to communicate with the backend that will do the word recreation for us. Overall, the project is currently on track for the deliverables we promised in the original proposal. There have been no major changes in scope or design to the project.