Meeting Global Needs for AAC

Cliff Kushler

Continuous Path Foundation

Brief History

1978: Began researching AAC in John Eulenberg's "Artificial Language Lab"

Early 90's: Co-invented original JustType with Martin King and Dale Grover as text input technique adapted for prototype eye gaze input with 8-cell selection matrix.

Late 90's: Commercialized JustType as "T9" and installed on over 8 billion phones ("predictive texting").

Early 2000's: Co-invented Swype with Randy Marsden. Later commercialized and installed on ~1 billion phones, and adopted or emulated on all major phone platforms.

2011: Created the Continuous Path Foundation to promote these technologies in the AAC field when Swype was acquired. Development efforts were resumed over the past two years resulting in:

2024: HeadBoard: Open-source Head-Tracking software that runs natively on Android devices using only the front-facing camera. Paired with "OpenBoard" (open-source Android keyboard IME supporting a Swype-like continuous path tracing input method). **Demo available in this session.**

2025: Re-designed JustType: Open-source, updated implementation of original JustType design including an UnDo/Delete key for easy recovery from keystroke errors plus integration with built-in head-tracking. **Demo available at this session.**

Continuous Path Foundation Mission Statement

CPF's Mission: Make effective AAC technology affordable world-wide.

Current Goals:

1. **World-wide:** Focus on making AAC devices and technology readily available in regions that have been under-served. This includes providing appropriate teaching and training resources.
2. **Effective:** Build on CPF's expertise in efficient text-based input methods (T9, JustType and Swype), and continue to design, develop and promote such methods in AAC.
3. **Affordable:** Create free, open-source solutions on low-cost platforms, and Android devices in particular, that enable the operator to use the entire device, rather than limit them to a single "AAC app."
4. **Practical:** Explore how text-based methods can be configured as tools for literacy acquisition. Integrate tools to teach literacy into symbol-based systems.
5. **Network** globally with individuals and organizations that serve the field of AAC, working together to advocate for and coordinate efforts to meet these goals.

Current Areas of Focus

Affordability:

- Android tablets and phones: Available everywhere; cheapest hardware platform (by far) with needed power and features. New tablets as cheap as $60 - $80!
- Developing free open-source software
- Wherever possible, minimize need for additional hardware by developing alternative software-only solutions

Support:

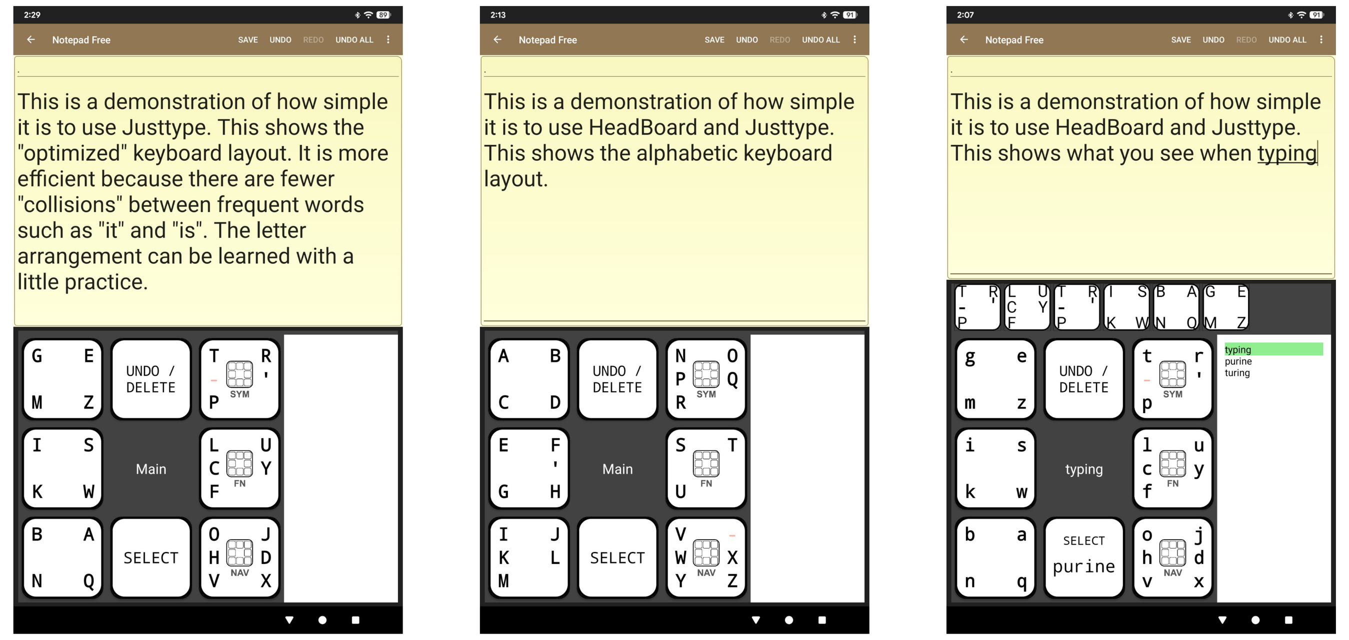
- Work with existing support organizations
- Try to develop ways to support consumers who might otherwise have little or no direct support
- Internationalization efforts: Core software and support materials

Design Goals for practical and effective solutions:

- Support a wide spectrum of input techniques to accommodate individual needs and changing abilities
  - **Critical Aspect: Reduce selection set to a *total of 8 keys***
  - **Implement selection techniques tailored for selecting from 8 keys**
- Support literacy acquisition:
  - Custom vocabulary management
  - Small keyboard facilitates integration with symbol-based systems
- Transparent UI: Enhance usability even in the absence of significant training

Let's connect!

I am actively seeking individuals who may be interested in trying out JustType and/or HeadBoard. I am also interested in exploring how a JustType keyboard can be integrated into a symbol-based device as a tool for literacy acquisition. Please contact me if you are interested!



Tablet screen-shots (roughly to scale)

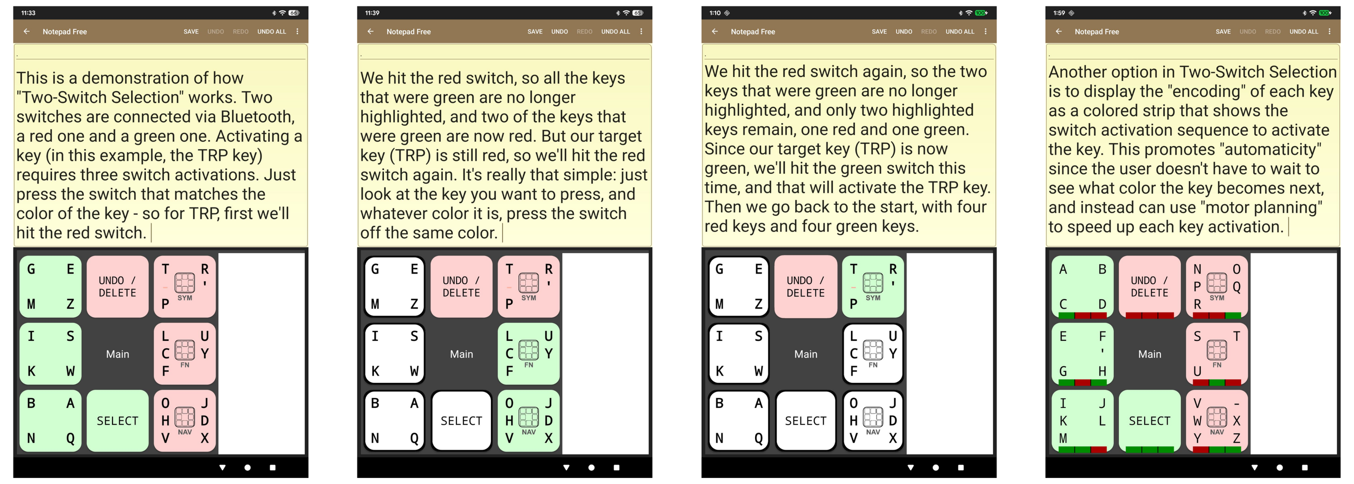


Illustration of Two-Switch Selection

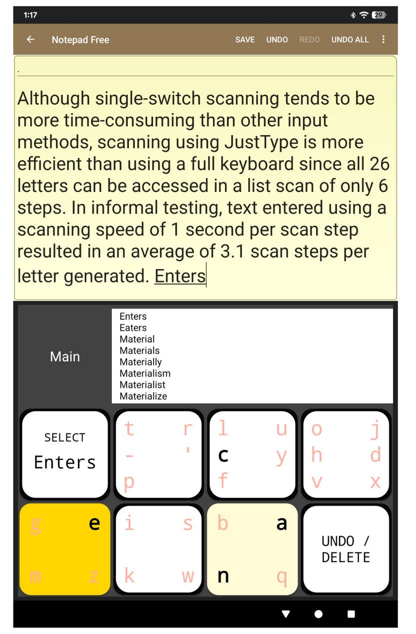


Illustration of Single-Switch Scanning

The screen shot on the left shows some of the optional features of single-switch scanning in JustType. One is that, as the letters of a words are entered (in the example shown, the word being typed is "entering") letters that do not occur in any word following the keys already entered can be "grayed out," making the "valid letters" (possible next letters) stand out visually. If this feature is enabled, the *scan can optionally "skip over"* any keys where all of the letters are grayed out, speeding up the scan. At left, the first six keys of the word "entering" have been entered. At this point, the TRP, OJHDVX, and ISKW keys are grayed out. At the instant of the screen capture, the scan had reached the GEMZ key (*dark yellow* highlight). Since the ISKW will be skipped over, the next key to be scanned (BANQ) is highlighted in *light yellow*, indicating that it is the *next key to be scanned*, and giving the user *time to anticipate* that the scanning switch will need to be activated on the next scanning step. This helps when the scanning pattern becomes less predictable.

Current JustType Features

Input Methods:

Direct Selection: Keys can be tapped directly on the touchscreen. This method can be simultaneously enabled with most other methods.

Head Tracking: Included "HeadBoard" software tracks head movement and converts motion toward one of the 8 "compass points" to key activations. The relative angular zone sizes of the "cardinal" (perpendicular) and "ordinal" (diagonal) directions can be adjusted since sometimes ordinal movements are more difficult to perform accurately.

External Joystick: Like Head Tracking, converts a joystick deflection toward one of the 8 "compass points" to a key activation.

Single-Switch Scanning: With only 8 keys, a linear scan is surprisingly efficient. Includes optional features to automatically scan the selection list, skip over "invalid" keys, and auto-repeat activations held the switch is held closed.

Current JustType Features (cont.)

Input Methods (cont.):

Two-Switch Selection: Enables efficient key activation with two switches with no timing requirements on switch activations.

"Directional" Selection: The touchscreen can be used to activate keys by stroking the touchscreen in the compass direction associated with the key. This gesture can be performed anywhere on the screen, so as long as a path can be traced in any of the 8 general compass directions, then direct selection becomes an option.

Eye-Gaze (future): Although eye gaze input on Android systems is not yet available beyond prototype systems offering two or three selections, it is reasonable to expect that eye-gaze input will become an option at some point. Again, since JustType requires selecting from only 8 keys, it will likely be one of the first practical options when the first eye-gaze implementations become available.

External Hardware:

- An external **Bluetooth joystick** is required for the Joystick input method. Although not yet tested, compatible wheelchair controller joysticks should be able to be configured to work.
- External **Bluetooth switches** can be configured for Single-Switch Scanning and Two-Switch Selection. Note that, when no external switch is available, the touchscreen can be configured as a single- or dual-switch, or a variety of facial gestures and/or head movements can be recognized and used as distinct switch inputs.
- When available, an external **Bluetooth speaker** can be used for voice output at higher volumes.

JustType Configuration Settings:

- Optional alphabetic layout arrangement. Note the following:
  - Although initially "strange" in appearance, the standard "optimized" layout will generally become automatic after a relatively short period of usage.
  - The alphabetic layout results in significantly more "collisions" between frequent words pairs: it/is, at/as, on/or/no, as/at, to/so, etc.
- Keyboard size adjusts to 40% - 95% of screen width
- Letters on keys can be "grayed out" when they do not occur in any word that matches the current key sequence entered.
- Output text (words, stored phrases, or completed sentences) can be automatically spoken.
- Activated keys and/or selected words can be spoken aloud to enhance accessibility by individuals with visual impairments.
- The key sequence entered can be optionally displayed as a row of key images.
- JustType supports ***abbreviation expansion***.
- Key activations confirmed by auditory (beeps) or visual (key flashes) feedback.

Why Use an 8-Key Keyboard? Pro's and Con's of JustType…

Con's: Additional cognitive load…

Pro's:

- *Physical effort required* to make each key selection is *drastically reduced*.
  - Additional benefits when the number of targets is a power of 2.
  - Each key selection can be ***much faster***.
- Ideal information theoretical "bang for the buck" of output/effort:
  - Claude Shannon estimated 2.62 bits of entropy ("information") per letter in a word. Choosing from a set of 6 keys requires 2.585 bits of information (3 bits for 8 keys). So a keypress in JustType is precisely **"worth the effort"** - you get what you pay for!
- Works with *high efficiency in a wide array of selection techniques*:
  - Joystick, SIngle-Switch Scanning, Two-Switch Selection, Head-Tracking (and facial gestures), Direct Selection, "Directional" Selection
  - Reduced selection set likely equally advantageous (still untested…) for BCI, Eyegaze, Voice, Other Techniques (?)
  - The same keyboard - the same ***interface to language*** - can continue to be used even if the selection technique needs to change due to a progressive condition (ALS, etc.)
- Includes *dedicated Delete key ("Undo") for simple error recovery* in all contexts
  - No overloaded, time-based key activations required (simplifies and speeds up BCI, Eyegaze, reduces switch activation issues)
- **Transparent UI**: Complete functionality is built into the 8 keys of the Main keyboard. Navigation to all functions and symbols is easily discoverable.
- JustType functions as the system keyboard. The ***entire device ecosystem becomes available***: texting, email, phone, document creation - everything Android offers
- Minimal screen space requirements - keyboard can ***co-exist with a symbol array***
  - Enables simultaneous keyboard animation to provide early literacy experiences while still communicating through symbols
    - When a word is selected, keyboard can be animated to show how to spell it
- Small keyboard *allows ample screen space for applications*

Important Next Steps for JustType

- Begin **extensive user testing** and receive feedback
- Work with literacy researchers and other developers to integrate JustType into one or more symbol-based Android AAC systems
- Dynamic recognition of creative spelling (should be feasible with small vocabularies)
  - Letters from alternative spellings can be *highlighted in a distinctive manner*
- Manage active vocabulary lexicon to help learn new vocabulary from "valid letter" prediction
- **Integrate LLM** to more quickly identify intended word from ambiguous keystrokes
  - Develop a model that self-evaluates confidence level of each prediction
  - Enable user to specify required confidence level to inject a prediction into the UI
  - Enable navigation to a predicted phrase mode for rapid conversation
- Support **navigation throughout the Android device** using the same 8-key selection technique used to operate the keyboard (provide alternative to HeadBoard)
- Port to **multiple languages**
  - Produce instructional support videos in multiple languages
- Try to engage industry partner to produce low-cost Bluetooth switches & joysticks
- Provide dual audio channels (public and private)

HeadBoard

Objective: Create open-source software to turn an inexpensive Android platform into a functional AAC system

Approach:

- Use a head-tracking cursor to enter text on a continuous path-tracing keyboard
- Also enable full control and access to the device; not just an "AAC app"

Goals:

- Try to create a usable AAC system based solely on an inexpensive Android device
- If possible, avoid requiring additional hardware
- Where additional hardware does improve functionality, seek an industry partner to produce

Progress:

- Started with "GameFace" open-source head-tracking software developed for Android
- Promising start, but significant usability issues made it difficult to accurately "Swype" (or sometimes even tap) text
- Used "OpenBoard" keyboard - open-source keyboard supporting continuous path-tracing input
- Technical issues have prevented using the more accurate GBoard available from Google. Attempting to resolve this.
- Made many enhancements to create very usable head-tracking. Next phase is testing with end-users.

Head-Tracking Enhancements in HeadBoard:

Goal: Improve accuracy and ease of use

- GameFace (internally, at least) has two "tracking modes" (my terminology):
  - "Absolute" - track movement of a particular point on the face in video frame
    - More "accurate" (linear response) but frequent need for recalibration
  - "Relative" - measure "pitch," "yaw," etc. of head orientation
    - Allows re-location of head in video frame, but response becomes non-linear
- Enhancement: Simultaneously use both modes
  - Absolute tracking of tip of nose for greater resolution/accuracy with linear response
  - Background automatic recalibration and centering with every X or Y zero-crossing in relative mode

Goal: Improve accuracy and ease of use

- Where possible and appropriate, divide screen into smaller "**Active Regions**"
- Biggest benefit: treat keyboard as separate region
  - Keyboard is where greatest accuracy is needed
    - Also very useful for Android navigation buttons, candidate selection displays, and text fields
  - Map entire range of head movement just to keyboard area
    - Higher resolution, greater control - the same absolute head movement maps to a smaller, more precise cursor movement
  - Whenever needed, crossing boundary into other regions is quick & easy - push against boundary for a settable delay

Result: Major enhancement to accuracy on keyboard

HeadBoard Enhancements (cont.)

Problem: Head-tracking does not provide a natural **"kinesthetic sense"** of where the cursor will go…

- "I see the cursor there, and I want it over there…" - but no intuitive sense of exactly *how much head movement will produce the desired result*; have to "try and see…"
- Because touch data is generated immediately wherever the cursor moves, the user must slow down to avoid overshooting and degrading Swype recognition with inaccurate touch data

Solution: Introduce a "**Path Cursor**" in addition to the existing "Head Cursor"

- *Path Cursor* shows the location where *touch data is actually being generated*
  - Head Cursor immediately reflects actual head position, but overshooting is not a problem; touch data comes from Path cursor and can be more heavily smoothed
  - Path Cursor appears in response to the signal to start generating touch data (switch closure, facial gesture, dwell, etc.)
  - Path cursor always moves toward current Head Cursor, but lags behind according to a user setting
  - User can quickly move Head Cursor toward target, often overshooting, but immediately reversing direction or adjusting as Path Cursor reaches target area

Goal: Intuitive, transparent, discoverable UI

- Android uses time-based distinctions to determine the intent of user actions: Ordinary tap, long tap, dragging, scrolling… These are distinguished by the duration of a touch action
  - In head-tracking, the user's attention is focused on the cursor
    - Use its appearance to convey how and when system state will change
  - Consistent association of colors with various states/actions: Green => continuous Swype; Yellow => Ordinary tap; Blue => Long tap; Red => Cancel; etc.
  - Introduce "**Cursor Animation** ": As an on-going action transitions from state to state, the cursor face is displayed with a "clock-sweep" animation from current color/action to the next. Example:
    - When cursor is *yellow* => stopping/moving away will perform an *ordinary tap*
    - After cursor animation completes clock-sweep to *blue* => stopping/moving away will perform a *long tap*
    - Settings adjust the length of delay time between states
    - *User can anticipate when to respond* in order to perform the desired action

Hardware Enhancement: Bluetooth Switch

- Avoiding the need for extra hardware is desirable, but… Accurate control of touch events is critical. Head tracking may solve cursor movement control, but there is still the need to control when a touch action begins and ends.
  - Dwell activation, facial gestures, etc., can be time consuming, tiring, or unappealing
  - Switch activation is easier, quicker, and generally more precise. Bluetooth switches can be positioned optimally for each user.
    - When controlling switch closure duration is challenging, a switch can be used in "Toggle On/Toggle Off" or "Tap" modes.
  - CPF has a limited quantity of prototype BT switch interfaces that work with standard "ability switches." Also commercially available for about $25.
    - To facilitate use when such hardware is unavailable, the touch-screen itself can be used as a switch. Better than nothing, but not ideal since device positioning can become complicated.

The screen shot at left shows HeadBoard being used to enter text by tracing a path on the OpenBoard keyboard. The green cursor is the "Head Cursor" that is responding in real time to the head movement seen in the camera view at the upper left. The blue cursor is the "Path Cursor." The Path Cursor constantly moves toward the Head Cursor, responding immediately to changes in direction, and as can be seen by the traced path at left, the touch data sent to the keyboard comes from the Path Cursor. This allows the user to move their head quickly (moving the Head Cursor quickly), often overshooting the target key, but with time to move toward the next target once the Path Cursor reaches the current target key.