Functionality vs Efficiency: A Comparative Study of Voice-commands and Touch Gestures as Navigation Systems in Android Video Games

Talayeh Amiri Tokaldany, Alice Chai, Adrian Fearman, Jeff Wang
Dept. of Electrical Engineering and Computer Science
York University
Toronto, Ontario, Canada M3J 1P3
tala9776@my.yorku.ca, chaia@my.yorku.ca, afearman@my.yorku.ca, jeffslw@my.yorku.ca

ABSTRACT

This research paper aimed to evaluate and compare the performance of two input methods, touch gestures and speech recognition, for a simple video game played on an Android device. The study found that there was no significant difference in system performance between the two input methods, indicating that they were equally effective. However, results of the study indicated that the user's satisfaction with the video game application was influenced by their preference for one input method (touch gesture) over the other (voice input). These findings suggest that developers should focus on providing users with a choice of input methods to ensure their satisfaction with the application.

Keywords

Android, speech recognition, touch gestures

INTRODUCTION

Stable and reliable speech recognition software has developed substantially over the last decade and has become one of the standard input methods for the majority of newly manufactured mobile devices [5]. This incorporation of speech recognition software into various applications available on the Android operating system has increased the familiarity of the technology among mobile device users. One of the user's primary interactions with this software is during employment of its speech-to-text functionality; using speech to complete a task that would alternatively require a device's built-in keyboard (e.g., the verbal dictation of a text message or email). Speech recognition technology is also commonly utilized when providing commands to virtual assistant applications or devices (in such cases as: Google's Google Assistant, Amazon's Alexa, and Apple's Siri).

The benefits of speech recognition as an alternative input method on mobile devices is that it increases the overall accessibility of the device. Users with disabilities, especially high incidence disabilities [7], are able to utilize speech as an alternative to standard text or touch input. These benefits are not restricted to users with disabilities as there are additional benefits in regards to ease-of-use, especially when considering users who may experience reduced technological literacy, such as in the elderly [6]. Furthermore, studies have been conducted which suggest that speech input (when compared to text input) can increase performance, cause less frustration, and prevent errors while also being less mentally, temporally, and physically demanding [3].

The limitations of some past studies is that comparisons were made between speech recognition and text input in terms of composition and transcription of language (e.g., comparing the speed in which a user can type a sentence using a keyboard versus dictating the sentence through speech recognition). These results are important, foundationally, to this study but the goal of this study is to further understand the ways that users can benefit from this type of technology.

In this study we investigated the ways in which speech input can benefit users beyond textual inputs. We evaluated the functionality and efficiency of using vocal commands while playing a platformer video game comparatively to the physical touch inputs using Android's built-in touchscreen capabilities. Beyond the user experience, the functionality of the device is also of great consideration in this study. In order to evaluate efficiency and functionality, the device's battery usage and overall performance (speed) was also considered when evaluating these input methods.

Related Works

With technology's gradual evolution towards a hands-free future, numerous researches have been conducted on speech recognition and voice commands; from speed comparisons with other input devices/systems, to efficiency and ease of use, speech recognition has been put into various tests throughout the past decade.

Studies comparing voice controls and other traditional controls have found that users preferred voice controls because it was more enjoyable and simpler to use [4, 8]. Uludağli and Acartürk [4, 8] compared the performance and engagement of users using gaze-voice command control (a combination of gaze and voice recognition) and touch control to play a simple platform game. They found that users who used the gaze-voice control method were more psychologically absorbed and immersed in the game. However, users were more successful (ie. less collisions with the obstacles) when playing the game

using touch controls [8].

Lee and Lai asked users to complete several e-mail and calendar tasks by interacting with a Mobile Assistant (MA) on the telephone by using Dual Tone Multiple Frequency (DTMF) input and speech input. The majority of users preferred using the speech input over the DTMF input. Users who preferred the speech input method liked its simplicity, intuitiveness, enjoyability to use, and how it was hands-free while users who preferred the DTMF input method thought it was less error-prone than speech. In regard to task performance, Lee and Lai found that DTMF input was more effective and efficient for linear tasks, while speech input was better for complicated and non-linear tasks [4].

Allison et al. used a revised version of Conway and Trevillian's SOC (Social/Operative/Character) model of the game event [2] to describe the player's experience with voice interaction gameplay. This revised model, referred to as the SFSC model, consists of four levels: the Social World, the Functional World, the Strategic World, and the Character World [1].

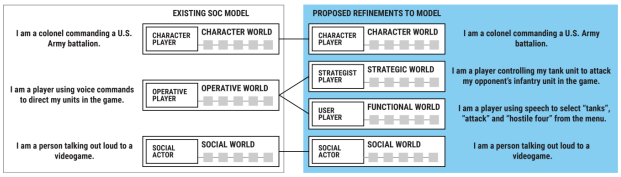


Figure 1. *SOC model and SFSC model of the game event [1].*

Users were asked to play three different voice interaction games with verbal, non-verbal, and manual controls. Allison et al. reported that users were more engaged in the game when using voice commands than when using traditional controls. Users described the game as more 'immersive' and 'real' and had a stronger sense of stepping into the role of the character. They also observed that while users were able to learn the functions of the voice commands quickly, they often mixed up the phrasing of the command (for example, "call tower" instead of "contact tower") [1].

COMPARATIVE STUDY

We conducted a comparative study of two navigation methods in a simple 2D endless runner video game, which can be played using either voice-commands or touch gestures. Our game, called *The Dino Game,* was inspired by Google's famous browser game, the *Dinosaur Game*, and features a dinosaur that endlessly runs across the screen from left to right. The game's objective is to jump over the obstacles by tapping or voicing a single word, in order to survive the endless run. To enable users to choose their preferred input method, we included two buttons on the game's main menu and as soon as they choose the input, the game will start (See Figure 2).

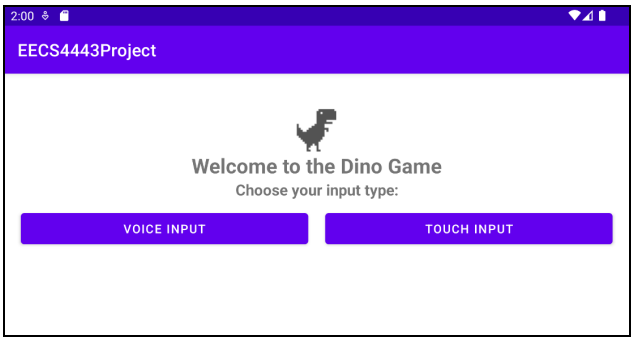


Figure 2. *The Dino Game's main menu.*

A timer located on the top-right corner of the game's main activity begins as soon as the game starts and keeps track of the player's survival time (See Figure 3), which is displayed on the "game over" screen when the dinosaur/player collides with an obstacle.

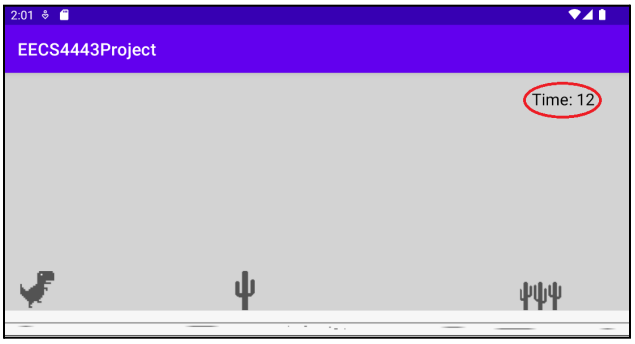


Figure 3. *The Dino Game's gameplay loop with a timer.*

From the "game over" screen, players can either try again or return to the main menu and switch input methods (See Figure 4).

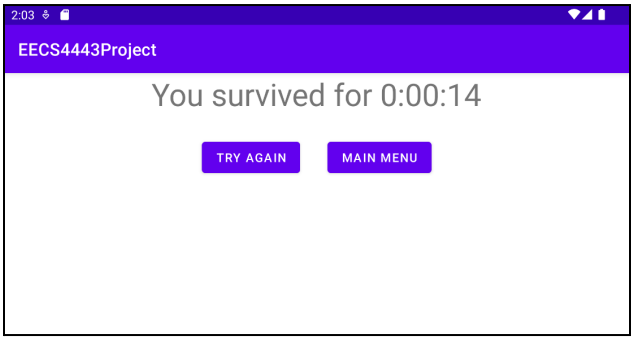


Figure 4. *The Dino Game's "game over" screen.*

The game can only be played in landscape mode, requiring the device's orientation to be horizontal.

We developed the game using Android Studio with Java as the primary programming language. For this version of the game, the touch gesture is "tapping" and the voice command is any single word spoken by the user throughout the game (for example: "jump"). The developers were the testers for this research, and used Android Studio's profiler to monitor the energy, CPU, and memory usage of the application.

Hypothesis Statement

The two input methods and their user interfaces were compared based on their speed, effectiveness, efficiency, and user satisfaction; *Effectiveness* is defined as how well a goal is achieved in a sense of absolute quality, *efficiency* as the amount of resources and effort that were used to achieve the specific goal, *satisfaction* as the degree that users were satisfied with a specific system [4], and their *speed* was determined by the time it took to complete a level successfully. A 2 x 2 comparative study design was conducted for a more thorough evaluation.

Based on other related works, we have two hypotheses:

H1: The input method (touch gestures or voice-commands) had no significant impact on the system's performance, it was just a matter of the player's preference.

H2: Based on previous research indicating touch gestures as the preferred method for simplistic tasks, touch gestures were expected to perform more efficiently over voice-command navigation in a simple endless runner game.

Apparatus

As mentioned earlier, *The Dino Game* was designed and implemented using Android Studio as the primary software.

Our application is compatible with Android smartphones and tablets. The devices must have at least one screen display, have an SDK level of 30, a screen resolution of at least 1080 by 1920, touch screen input capability, and a voice input system.

To evaluate the performance and energy consumption of our application, we used the Android Studio Profiler. This allowed us to monitor the energy, CPU, and memory usage of *The Dino Game* during testing.

Procedure

To assess the performance of the application, Android Studio's built-in profiler was used to monitor the CPU, memory, and energy usage for both touch gesture inputs and voice-command inputs. Automated tests were considered in order to test the execution of the application and to help evaluate the system performance. However, at the time of this research, there were no ways to test voice-commands with automated testing tools such as Espresso Therefore the system performance was monitored as the tester played the game in real-time.

Incremental testing was done to test the application as it was being developed. We used both bottom-up and top-down testing methods; bottom-up was used when testing our back-end and code, while top-down was used when testing our front-end and interactions.

Design

For our application, we conducted a 2 x 2 comparative study design using automated test scenarios. Our application is designed as a 2 x 2, since it was developed to examine the impact of 2 independent variables with 2 different levels on the dependent variables.

The two independent variables for our study are:

- Input method (voice-commands, touch gestures)
- Settings (phone, tablet).

The dependent variables are the performance and battery consumption metrics, which include speed of task completion, effectiveness, efficiency (CPU, memory, network, battery usage), which are also used to infer user satisfaction.

The number of experiments in this study is 2 x 2 = 4.

RESULTS AND DISCUSSION

Test Cases

The following is a collection of all UI test cases. VITest1 and TITest1 were the test cases used for this study's profiling results. VITest2 and 3, and TITest2 and 3 were used for testing UI elements separately from the main test cases, but not for profiling.

Voice Input Test Cases

Test Script ID	VITest
Title	Voice Input Game Test
Test Case ID	VITest1
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Number of jumps (5) 4. Timer 5. Input (voiceListener -> VoiceInputActivity)
Procedure	1. User launches the app. 2. From the main page, user clicks the 'Voice Input' button 3. Game initiates. 4. As the obstacles approach, the user says "jump" to cause the main character to successfully jump over the obstacles. 5. This is repeated a total of 5 times. 6. On the 6th obstacle, the user says nothing and the main character runs into the obstacle. 7. The game ends, the user is taken to a game over screen.
Actual Result	1. The user's score is displayed successfully (in time) on the top of the game over screen. 2. The user is presented with two options: restart the game by clicking the Try Again button, or click the Main Menu button to return home.

Status	1. ResultsActivity is successfully launched upon running into the obstacle. 2. In the ResultsActivity the correct score is shown at the top of the View. 3. The user is presented with two clickable buttons (Try Again and Main Menu) in the ResultsActivity
Defect Cross-Reference	No defect detected

The results of VITest1 are further discussed in the results section of this paper as it was used during the profiling stages. The results of this test case indicate that the game is playable in the voice input mode. No defects were detected upon multiple tests. Test is repeatable but score results may vary between 8 and 9 seconds as distance between obstacles is not uniform, offering variability between game plays.

Test Script ID	VITest
Title	Voice Input Select Test
Test Case ID	VITest2
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Timer 4. Input (voiceListener -> VoiceInputActivity)
Procedure	1. User launches the app. 2. From the main page, the user clicks the 'Voice Input' button. 3. Game initiates in the correct mode (Voice Input).
Actual Result	1. Although the mode is not visible to the user, upon clicking the Voice Input button, the game launches in the Voice Input mode (VOICE_INPUT) 2. Game launches VoiceInputActivity
Status	1. Game is launched in VOICE_INPUT mode. 2. Touch input is disabled. 3. Voice input is enabled.
Defect Cross-Reference	No defect detected

The results of VITest2 indicate that the game correctly launches in the correct game mode when the 'Voice Input' button is selected from the main menu. This test is repeatable with unvarying results and no defects. This result is a quintessential element in the functionality of the game as the user's expected input must match the selected game type at the beginning of each game.

Test Script ID	VITest
Title	Voice Input From Results Test
Test Case ID	VITest3
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Timer 4. Input (voiceListener -> VoiceInputActivity)
Procedure	1. User has lost a game while playing in Voice Input mode and is on the ResultsActivity View. 2. The user is prompted with two buttons. One that says 'Try Again' and the other that says 'Main Menu' 3. User Selects 'Try Again' 4. Game is launched again using the Voice Input mode.
Actual Result	1. Although the mode is not visible to the user, upon clicking the Try Again button, the game launches once more in the Voice Input mode (VOICE_INPUT) 2. Game launches VoiceInputActivity
Status	1. Game is launched in VOICE_INPUT mode. 2. Voice input is enabled. 3. Touch input is disabled. 4. Repeatable multiple times.
Defect Cross-Reference	No defect detected

The results of VITest3 indicate that the game is correctly relaunched when the user selects the 'Try Again' button from the ResultsActivity view. By correctly relaunched we mean that the game is launched once more in the same mode that was initially selected from the Main Menu. This test is repeatable with the same results without any defect detected.

Touch Input Test Cases

Test Script ID	TITest
Title	Touch Input Game Test
Test Case ID	TITest1
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Number of jumps (5) 4. Timer 5. Input (touchListener -> TouchInputActivity)
Procedure	1. User launches the app. 2. From the main page, user clicks the 'Touch Input' button 3. Game initiates. 4. As the obstacles approach, the user touches anywhere on the view to cause the main character to successfully jump over the obstacles. 5. This is repeated a total of 5 times. 6. On the 6th obstacle, the user does not touch the view and the character runs into the obstacle. 7. The game ends, the user is taken to a game over screen.
Actual Result	1. The user's score is displayed successfully (in time) on the top of the game over screen. 2. The user is presented with two options: restart the game by clicking the Try Again button, or click the Main Menu button to return home.
Status	1. ResultsActivity is successfully launched upon running into the obstacle. 2. In the ResultsActivity the correct score is shown at the top of the View. 3. The user is presented with two clickable buttons (Try Again and Main Menu) in the ResultsActivity
Defect Cross-Reference	No defect detected

The results of TITest1 are further discussed in the results section of this paper as it was used during the profiling stages. The results of this test case indicate that the game is playable in the touch input mode. No defects were detected upon multiple tests. Test is repeatable but score results may vary between 8 and 9 seconds as distance between obstacles is not uniform, offering variability between game plays.

Test Script ID	TITest
Title	Touch Input Select Test
Test Case ID	TITest2
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Timer 4. Input (touchListener -> TouchInputActivity)
Procedure	1. User launches the app. 2. From the main page, the user clicks the 'Touch Input' button. 3. Game initiates in the correct mode (Touch Input).
Actual Result	1. Although the mode is not visible to the user, upon clicking the Touch Input button, the game launches in the Touch Input mode (TOUCH_INPUT) 3. Game launches TouchInputActivity
Status	1. Game is launched in TOUCH_INPUT mode. 2. Voice input is disabled. 3. Touch input is enabled.
Defect Cross-Reference	No defect detected

The results of TITest2 indicate that the game correctly launches in the correct game mode when the 'Touch Input' button is selected from the main menu. This test is repeatable with unvarying results and no defects. Like the VITest2, this result is a quintessential element in the functionality of the game as the user's expected input must match the selected game type at the beginning of each game.

Test Script ID	TITest
Title	Touch Input From Results Test
Test Case ID	TITest3
Data	1. Game Type (voice activated or touch) 2. Number of obstacles (6) 3. Timer 4. Input (touchListener ->

	TouchInputActivity)
Procedure	1. User has lost a game while playing in Touch Input mode and is on the ResultsActivity View. 2. The user is prompted with two buttons. One that says 'Try Again' and the other that says 'Main Menu' 3. User Selects 'Try Again' 4. Game is launched again using the Touch Input mode.
Actual Result	1. Although the mode is not visible to the user, upon clicking the Try Again button, the game launches once more in the Touch Input mode (TOUCH_INPUT) 2. Game launches TouchInputActivity
Status	1. Game is launched in TOUCH_INPUT mode. 2. Voice input is disabled. 3. Touch input is enabled. 4. Repeatable multiple times.
Defect Cross-Reference	No defect detected

The results of TITest3 indicate that the game is correctly relaunched when the user selects the 'Try Again' button from the ResultsActivity view. By correctly relaunched we mean that the game is launched once more in the same mode that was initially selected from the Main Menu. This test is repeatable with the same results without any defect detected.

Profiling

The Dino Game was profiled using a Samsung Galaxy Note20 Ultra running Android 12.0. An eight-second run, with five jumps, was done with each input method, and the CPU usage, memory allocation, and energy consumption were monitored using Android Studio's profiler. It was found that there is no significant difference between the performance of the two input methods (See Figure 5).

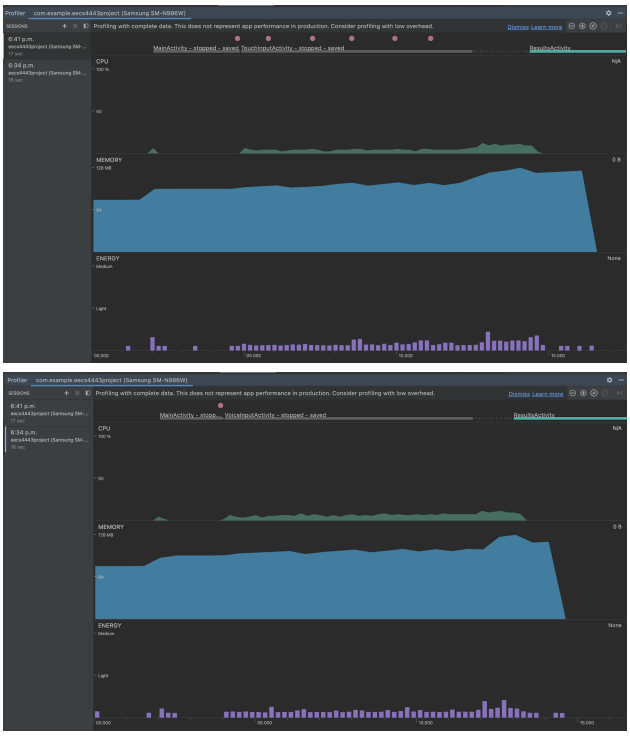
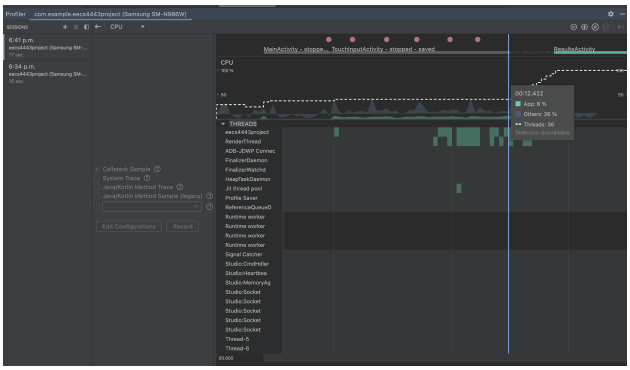


Figure 5. *Touch input (top) vs voice input (bottom) performance.*

The Dino Game's CPU stayed in 0-1% if no actions were performed. Spikes in CPU usage appeared whenever the user interacted with the app (ex. when the user tapped on the screen or shouted to jump). The majority of CPU usage was by other threads rather than the user's interactions. Voice input used the CPU more consistently throughout while touch input seemed to use the CPU only when the user tapped on the screen hence the difference in the shape of the usage graphs (See Figure 6).



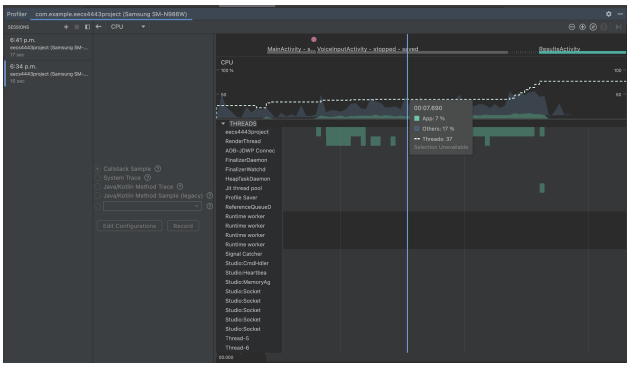


Figure 6. *Touch input (top) vs voice input (bottom) CPU usage.*

The Dino Game's memory was consistently between 70-90 MB without user actions. Spikes in memory usage appeared whenever the user interacted with the app (ex. when the user tapped on the screen or shouted to jump), with a larger spike when the user runs into an obstacle and the ResultsActivity is loaded (See Figure 7).

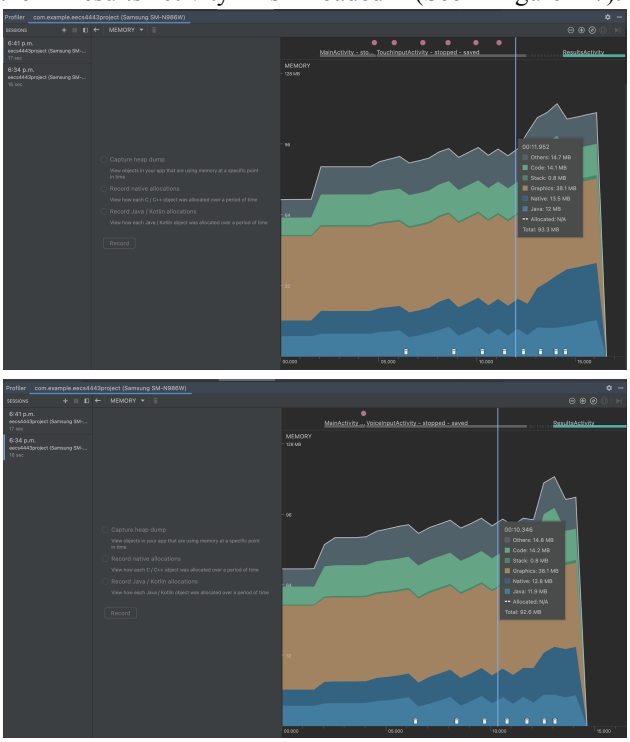


Figure 7. *Touch input (top) vs voice input (bottom) memory usage.*

The app's energy usage stayed light throughout.

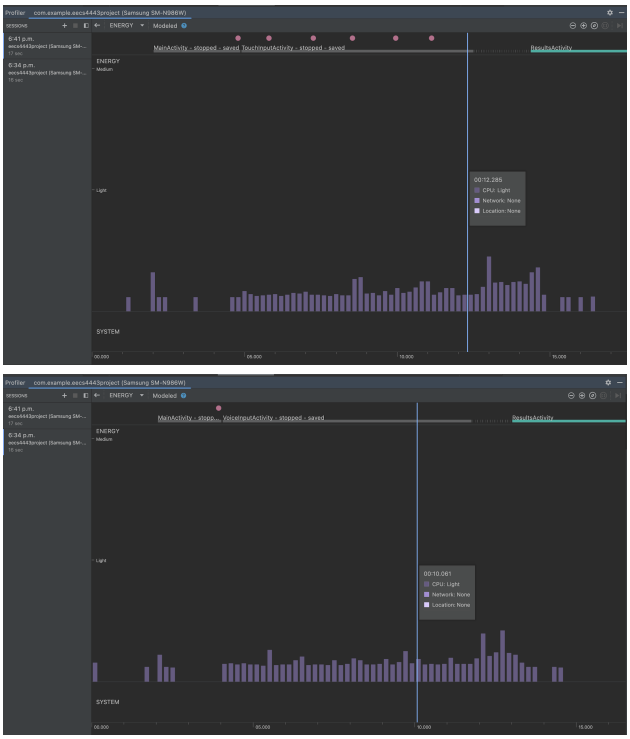


Figure 8. *Touch input (top) vs voice input (bottom) energy usage.*

CONCLUSION

The goal of this study was to determine if voice input had any significant impact on user experience and/or system performance when compared to touch input while running a game. A simplistic game and corresponding UI were designed to make the comparative tests simple and effective and to allow for increased accessibility. The simple design included a game that did not contain elaborate rules or instruction, and a UI system that offered the user easy selection between the two possible modes (voice input, touch input). The UI was tested using various test cases, as described above in the Results and Discussion section of this paper. During this UI testing no defects were detected which would impact the profiling of either versions of the application. A larger test, which included a full run of the application from main menu to results screen (after the player had lost) was used for profiling purposes in order to determine if there were observable differences in either input mode. The test variables were kept the same under both conditions with the same number of obstacles and the same number of 'jumps' (committed either with a vocalization in the voice input mode, or with a tap in the touch input mode). After running the tests repeatedly, the results of the performance tests concluded that there is no significant difference in CPU, memory, or energy usage between the two types of input. This is significant because any increase in performance usage could lead to disruption in the user's experience within the game activity. Observationally, this corresponded to no delay or load time differences.

Qualitatively, the results of this study found that the voice input mode of this game required more concentration and coordination in order to perform successful 'jumps'. A reason for this could be that the users who performed these actions were not used to voice activated input for a game of this nature. This is further supported as the users who tested the voice input method became more successful over time, implying a learning curve for this type of input. More research would need to be done specifically in this area; ideally with two groups of test subjects, those who have experience with voice activated game software and those that do not. What can be concluded about this study is that a voice activated version of a game is a way to increase accessibility within mobile applications without sacrificing processing power or UI simplicity and structure. Although a separate input mode had to be developed, the core functionalities of the game processing and logic remained the same and there was no observable impact on the functionality of the device at play time. This study can be used to argue that adding voice activated features to increase accessibility within an application or game is justified.

REFERENCES

1. Allison, F., Newn, J., Smith, W., Carter, M., & Gibbs, M. (2019). Frame analysis of voice interaction gameplay. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3290605.3300623
2. Conway, S., & Trevillian, A. (2015). "blackout!" unpacking the black box of the game event. *Transactions of the Digital Games Research Association*, *2*(1). https://doi.org/10.26503/todigra.v2i1.42
3. Foley, Casiez, G., & Vogel, D. (2020). Comparing Smartphone Speech Recognition and Touchscreen Typing for Composition and Transcription. Conference on Human Factors in Computing Systems - Proceedings, 1–11. https://doi.org/10.1145/3313831.3376861
4. Lee, K. M., & Lai, J. (2005). Speech versus touch: A comparative study of the use of speech and DTMF keypad for navigation. *International Journal of Human-Computer Interaction*, *19*(3), 343–360. https://doi.org/10.1207/s15327590ijhc1903_4
5. McGraw, Ian et al., "Personalized speech recognition on mobile devices," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 2016, pp. 5955-5959, doi: 10.1109/ICASSP.2016.7472820.
6. Ogozalek, & Van Praag, J. (1986). Comparison of elderly and younger users on keyboard and voice input computer-based composition tasks. SIGCHI Bulletin, 17(4), 205–211. https://doi.org/10.1145/22339.22372
7. Ok, M. W., Rao, K., Pennington, J., & Ulloa, P. R. (2020). Speech recognition technology for writing: Usage patterns and perceptions of students with high incidence disabilities. *Journal of Special Education Technology*, *37*(2), 191–202. https://doi.org/10.1177/0162643420979929
8. Uludağli, M. Ç., & Acartürk, C. (2018). User interaction in hands-free gaming: A comparative study of gaze-voice and touchscreen interface control. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, *26*(4), 1967–1976. https://doi.org/10.3906/elk-1710-128