# CodeBash UserManual
# CSCI 205 - Fall 2022 - team05

**Problem Statement -** *What we aim to solve*

Technological advancements and the growth of the tech industry has increased the amount of time individuals spend in front of screens. Most professions require technological competency as a key characteristic for employment. With productivity being ever more dependent on the efficiency of our interactions, the most productive employees have become those that can optimize their interactions with their devices. That is where CodeBash comes into play. We aim to optimize the most important way we interact with our computers, through typing.

A faster typing speed reduces fatigue, increases productivity, and allows for the creation of  new ideas as it allows the brain to remove focus from the actual keys being typed. According to an article written by North Central College*, Is Computer Science Hard ?* , the average computer science student spends around 18 hours a week (North Central College. (2022))  in front of a computer. That time can be reduced with a faster typing speed allowing them to spend more time on their other assignments or themselves. Another problem that CodeBash aims to solve is switching coding languages. Developers, throughout their work, will encounter times where they might have to code in a different language. The problem comes when they forget  the syntax and need to familiarize themselves. CodeBash offers a solution by creating a space to practice syntax of different languages in a simple and fun format. This saves time and energy as they avoid having to look at old reference material.

Is Computer Science Hard? | North Central College. (2022). Retrieved 6 December 2022, from https://www.northcentralcollege.edu/news/2021/03/24/computer-science-hard

**Introduction**

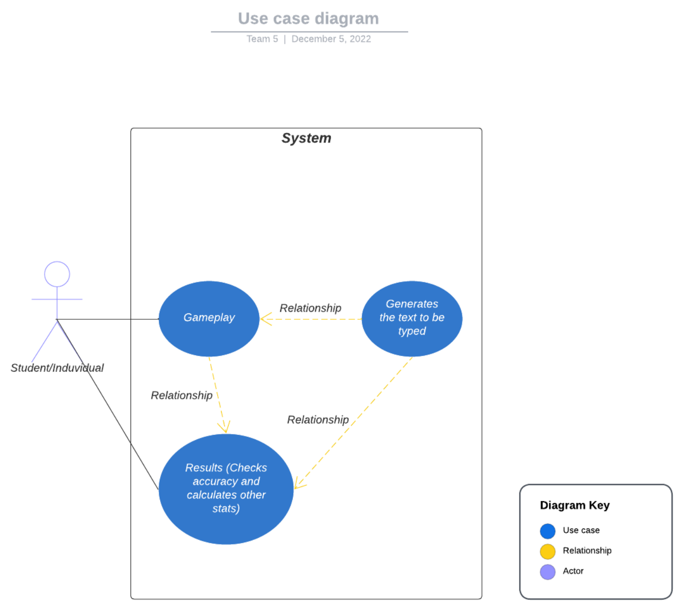


Fig. 0 - The Use case diagram

The challenge of creating a reactive, customized typing game comes from three main areas: human interaction, display, and the logic to complete analysis. We needed to design a system that was functional on multiple devices, meaning it works on a mac, linux, windows, etc... This in itself is a big challenge because different computers handle keyboard input differently thus it's ever more important to have completed exhaustive testing. As displayed in (Fig.0) our use case diagram the main point of interaction is with the user thus handling keyboard input is especially important. Furthermore, in order to optimize the user experience we had to make sure we had a pleasant display. The ***monkeytype*** game was a huge source of inspiration as they were able to create a smooth and appealing interface. Lastly, the logic of handling user input, checking for accuracy, and calculating speed statistics was another area we had to do extensive research on. This is an important part of our game since it's what provides feedback and allows our users to utilize it for self improvement.

## Background

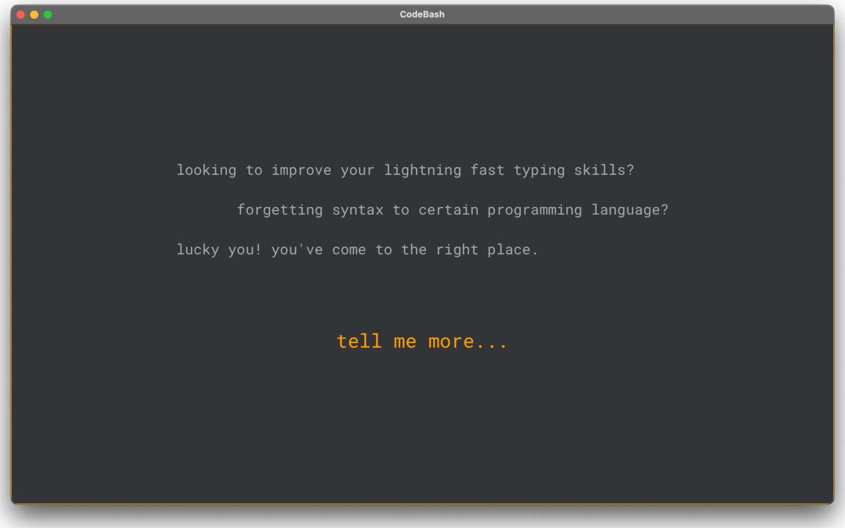
The research to solve these main pain points spanned in many different directions. To begin, to create a system that works on multiple devices we constantly tested the game play on each of our devices: Mac, Linux, Windows, and Gaming laptops. What we saw was that depending on how we handled the keyboard input, some characters would not be registered on the screen. The main keys that gave us trouble were the shift and space keys. With the help of *stack overflow* and the java library we were able to solve this problem by implementing a general 'setOnKeyPressed()' method. This, as we saw from testing, was able to read input on all of our devices. Furthermore, to make the appearance of our game as similar and as smooth as the one for the popular game ***monkeytype,*** we looked at their javaScript source code on *github.* We were able to gather the exact color schemes they used and implement them into our code. This allowed us to reach the quality of appearance that we were aiming for. Lastly, the logic part of our code was an area we had to improvise and derive words per minute calculations from other realizations of this similar game. However, we did have to make a solution that was able to adapt to the character by character method of handling input. This came with its own problem as the way we handled the typed sets of words was entirely dependent on pressing the spacebar. When it comes to the dictionaries we used to create our vault of possible text displays, we derived the Idea of scanning text from previous labs and we custom created the coding dictionaries using references we found from ***monkeytype.***

## Motivation

When we initially started, we had wanted to create the game Sudoku. We wanted to create Sudoku and solve the problems but at the same time, have an ability for the randomly generated problems to solve itself. Then we decided to switch to a typing game that also can be used to type in the coding styles and not just sentences in English. The motivation behind this game is to transform the traditional typing game for those interested in computer science. By allowing more functionality and unique themes, we are able to improve user experience and typing development. We also understand that we aren't the first ones to create a typing game thus improving on existing versions allows us to make a major contribution.

## Instructions - *How to Play CodeBash*

After running CodeBashMain or typing **./gradlew run** in the terminal, the user is met with an Introduction Scene (Figure 1), meant to spark interest and give them a greater context into what this game was made for.



CodeBash

looking to improve your lightning fast typing skills?

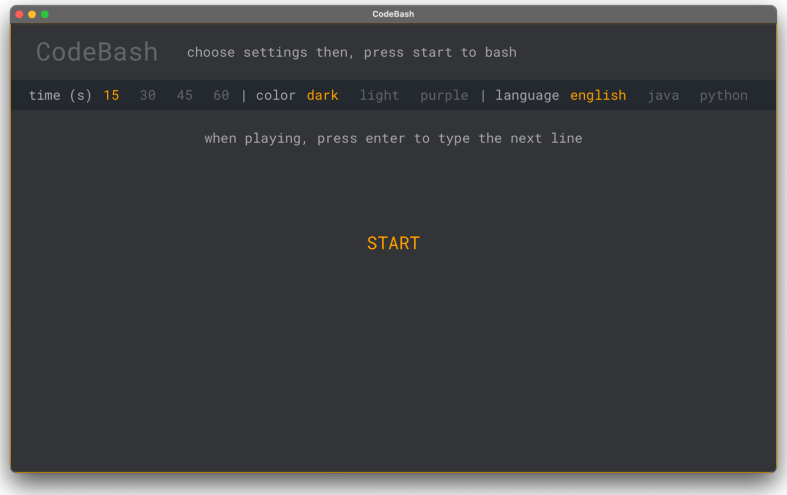
forgetting syntax to certain programming language?

lucky you! you've come to the right place.

tell me more...

Fig. 1 - The Introduction Scene

Then, **click the yellow text button "tell me more…"**, to bring you to the Welcome Scene (Fig. 2). This where the user can **choose settings** for the game, shown in the darker colored bar at the top of the screen.



CodeBash

CodeBash    choose settings then, press start to bash

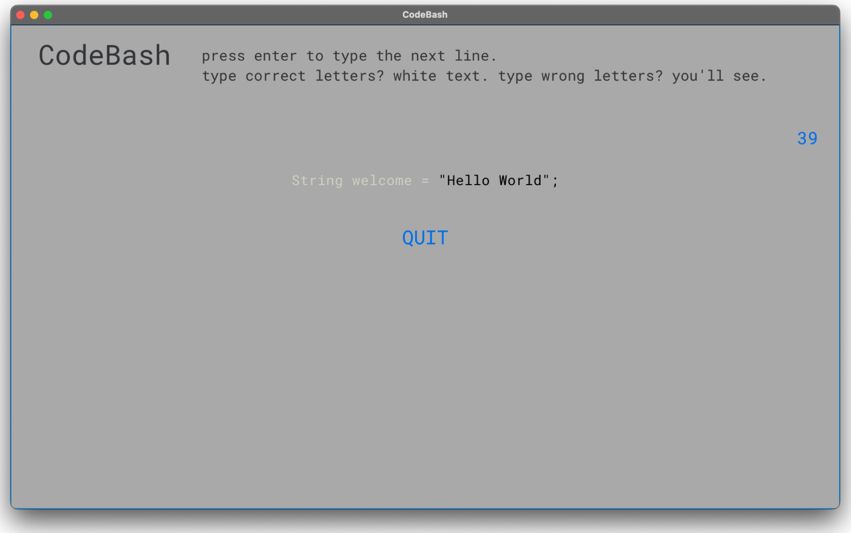
time (s) 15  30  45  60 | color dark  light  purple | language english  java  python

when playing, press enter to type the next line

START

Fig. 2 - The Welcome Scene

The default settings, as shown in Fig. 2, are 15 seconds of game play, dark mode, and english. From left to right, the user can choose an amount of time to play the game for (15 seconds, 30 seconds, 45 seconds, or 60 seconds). When the user clicks any of those buttons, the setting is applied and the button turns orange (when in dark mode) to show it was selected. This holds for the other settings. Next, the user can choose a color mode (dark mode, light mode, or purple mode). This setting applies the color theme across any of the future scenes the user interacts with. Lastly, the user can choose a language to type in (english, java, or python). Once the user is happy with their choices, the user can **click "START"** to begin game play.



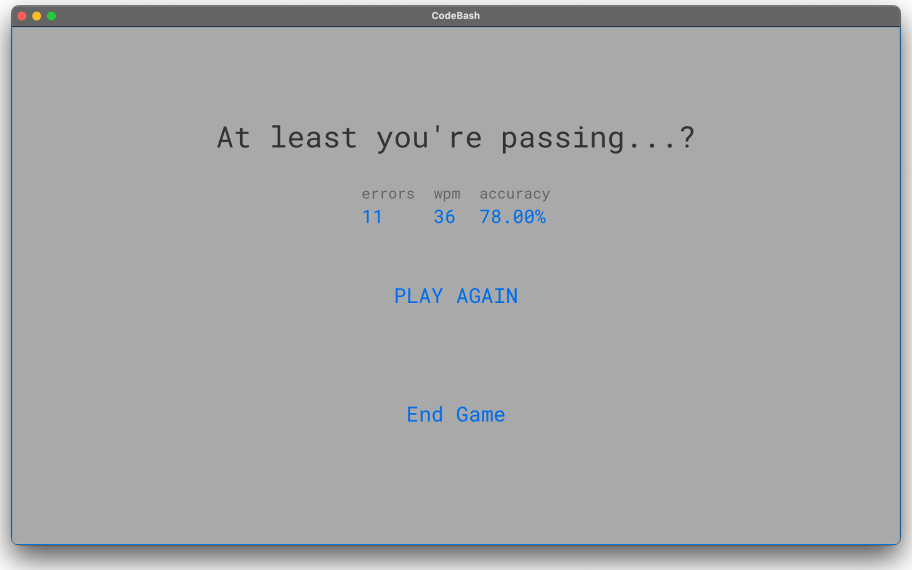
```
press enter to type the next line.
type correct letters? white text. type wrong letters? you'll see.

39

String welcome = "Hello World";
```

Fig. 3 - The Welcome Scene

In Fig. 3, this is an example of the user picking 45 seconds, "light mode", and java for their game play settings. To play the game, the user has until the timer stops to type as much text as accurately as possible. The text changes to white text when the user types correctly. The text changes to the highlight color, which in light mode is blue, when the user types incorrectly. To move to the next sentence, **press Enter**. To end the game, **press "QUIT"** or **wait for the timer to end** while typing.



At least you're passing...?

```
errors  wpm  accuracy
11      36   78.00%
```

Fig. 4 - The Results Scene

Then, the user is brought to a Results Scene shown in Fig. 4. Starting from the top, the user is displayed a unique statement describing how they played. For instance, the user was typing slowly but accurately, so the game snarkily summarized it for the user with "You're putting me to sleep with that speed…". Below the text feedback are statistical representations of the user's game play, including the amount of errors they made (11), words per minute (36 wpm), and accuracy (78.00%). To play again, the user can **click "PLAY AGAIN"**. The user is then **brought back to the Welcome Scene** (Fig. 2), to choose settings and continue playing. To end the game completely and close the application, the user can **click "end game"**.