Click It: A Game By Phillip, Gavin, and Zoha
User Manual

Table of Contents:

How To Play

- Click It is a JavaFX game that consists of two windows: NewGamePage.java and MainGame.java
- Make sure all game files (NewGamePage.java, MainGame.java, DatabaseClass.java, NewGameStyles.css, MainGameStyles.css, and PlayerDB.sqlite) are in the same directory and launch the game by compiling and running NewGamePage.java.
- Upon launch, the user is brought to a screen where the top 3 scores are displayed, underneath which they are prompted to enter their name and click the 'PLAY' button. Additionally, the user can click the 'INSTRUCTIONS' button to display a condensed version of this page (the rules).
- If the name entered (case-insensitive) already exists in the PlayerDB database (returning player), the already-existing row in the database with the entered name becomes associated with the current player. If the name entered DOES NOT exist in the PlayerDB database when it is entered, a new row with that name will be added to the database.
- After the play button is clicked, the game is launched (MainGame.java).

Rules of the Game

- The game consists of 3 rounds with 10 screens per round
- In round 1, each screen counts down from 2.5 seconds, in round 2, each screen counts down from 2 seconds, and in round 3, each screen counts down from 1.5 seconds.
- Once the screen timer has reached 0, the screen number is accumulated and when the screen number is 10, the next time it is accumulated, it accumulates the round number, until round 3, screen 10 is passed, marking the end of the game.
- The game displays a grid of 9 buttons - 8 read "No", while 1 reads "It's Me". Each time a screen timer reaches 0 AND each time any button is clicked, the buttons are rearranged.
- The objective of the game is to click the "It's Me" button as many times as possible before the end of the game.
- If the "It's Me" button is clicked in round 1, the user earns 5 points. If they click the right button in round 2, the user earns 10 points. If they click the right button in round 3, the user earns 20 points.
- At the end of the game, the user's final score is displayed in the score display.
- At any point in time, 'END GAME' can be clicked to return to the New Game Screen
- *Do you have what it takes to make it into the top 3?*

NewGamePage

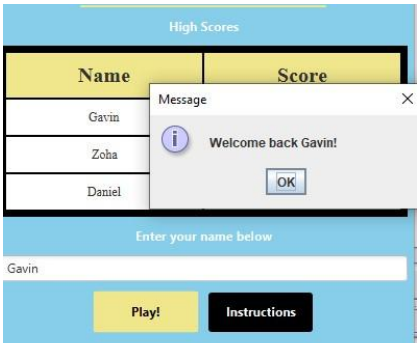
This window is the first window the user will use before they start the actual game. The player will have to type their names and click the play button to get to the MainGame window, after playing the game which is in the **MainGame** window, this NewGamePage will pop up and if the player has a higher score than the previous player(s), then the current player's name will be shown in the top 3 list.

Name	Score
Gavin	250
Zoha	200
Daniel	150

Some details with screenshots:-

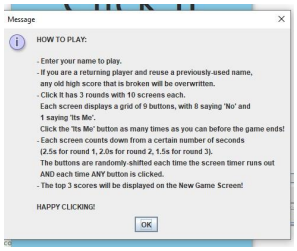
The image to the left shows the top 3 players who had the highest scores. So if a player gets a higher score than Gavin (like 260) they will replace Gavin; Gavin will replace Zoha, and Zoha will replace Daniel, and Daniel will no longer be in the top 3.

Important think to note when a player is typing their name there should not be any spaces because if the player do have spaces then the screen will display, "ERROR! Please enter a name with no spaces" as shown in the image to the right.



If the current player had played the game before, a window will display saying welcome back, as shown because the program recognizes the name of player already played the game.

If the player is new, the screen displays just "Welcome" and their name.

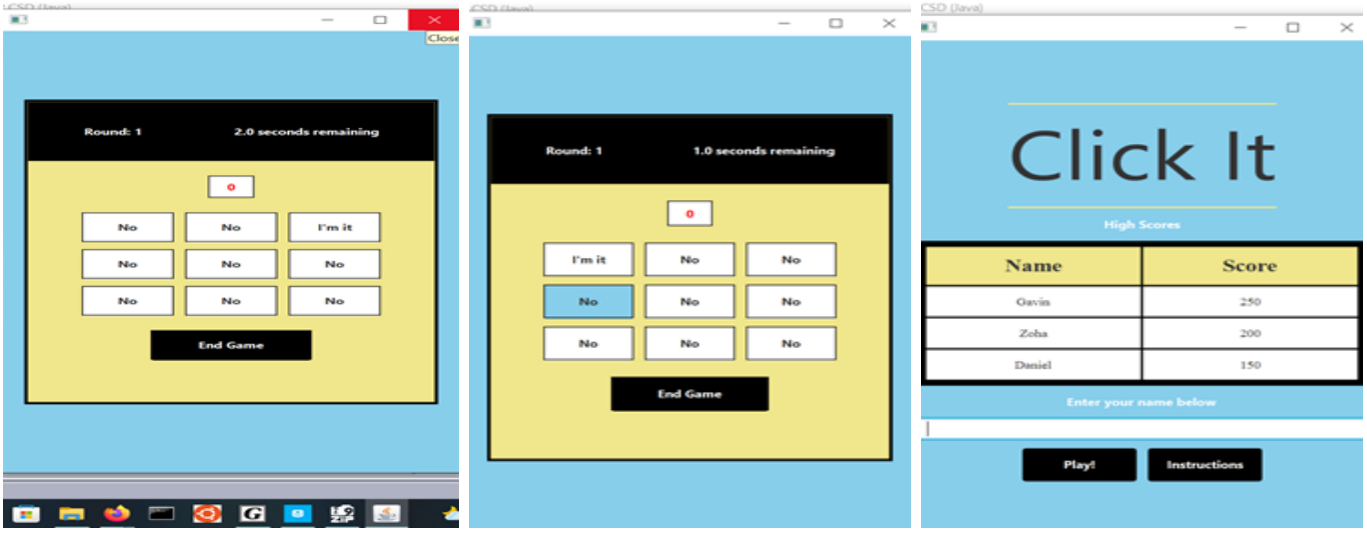


When the NewGamePage window shows up, there is a button next to the "play button" named "instructions" when the player press this button a message window pops up giving some basic instruction about how the game works.

If the current player had played the game before, the player can enter their name in any combination of lower and upper case and the program will still recognize their name. As shown in the image to the left, when typed "daniel" the program still recognize it as "DANIEL".

MainGame

The window that the actual game takes place in, after clicking the 'PLAY' button on the **NewGamePage** window, is **MainGame.java**. **MainGame** displays a window with three primary sections: Up top is the *display section,* which displays the round number, the screen timer, and the current score. In the middle of the window is the *button grid*. At the bottom of the page is the *'END GAME' button*. Below, the workings of the code are explained in detail.



MainGame.java is launched by the 'PLAY' button on the **NewGamePage** window. Upon launching, the game is started, starting at round 1, screen 1, and it goes on from there until it reaches the end of the game after round 3, screen 10..

Each time a grid button is clicked, or a screen timer reaches 0, a new grid of buttons is generated, with the "I'm It" button being randomized each time. This is accomplished by making the event handler of the grid buttons create a new 3x3 array of buttons, generate a new button object for each index in the array, randomly-assign one of the buttons the text "I'm It" and all others "No", and save the indeces of the "I'm It" button each time a grid button is clicked. The timer works by utilizing Java's **Timeline** and **KeyFrame** classes. A **Timeline** can be thought of as a *sequence of animation frames*, like in a video game. The **cycle count** (how many KeyFrames are cycled through in the Timeline) is set to *10 times whatever the round time is* (round 1: 2.5s, round 2: 2.0s, round 3: 1.5s) and the duration of each frame is set to 0.1 seconds. Each time a KeyFrame cycle is run, the time left is decremented by 0.1. The game logic is set to end when the round number is no longer <= 3, so after round 3, screen 10, the game ends. Once that happens, the program checks to see if the player's current score is greater than their high score in the database. If it is, their database score is overwritten. If their score is in the top 3, it will now be displayed on **NewGamePage.java**. The following is a step-by-step description of how the program runs. When the 'PLAY' button is clicked and a new instance of **MainGame** is launched, the name that was entered into the TextField is passed into the constructor of **MainGame** in order to save the player's name for database access. Next, the controls and layout containers are initialized. Then, the initial button grid is generated and the *countdown method* starts. The countdown method works by using the classes described above to implement a **recursive loop**. When the countdown method is called, it checks to see what the current round number and screen number are, and then it uses the **Timeline** and **KeyFrames** specified above to count down from the time for round 1 (2.5s). *Then, while the round number is <= 3, the countdown method calls itself*. This creates a loop where each time the countdown method is called, it checks to see what the current round number and screen number are. It then sets the cycle count to 10 times whatever the round time is, sets the timer's max time to the time corresponding to the round number, and

then, using **KeyFrames** and decrementing round time by 0.1 each frame, the timer counts down from that max time. Once round 3, screen 10 is passed, the loop stops and the game ends. The program then checks the player's database score and overwrites it with their current score if they beat it. If, at any point, the 'END GAME' button is clicked, the current game will be discarded and the user will be returned to **NewGamePage**.

DatabaseClass.java

The DatabaseClass makes use of five methods to read, insert, and update the database in PlayerDB.db:

- getTopPlayers: This method is called in NewGamePage, which passes the labels to hold the names and scores of the players. Inside getTopPlayers, a statement will be created, as well as a string to hold the query that will select all data from the Players table in PlayerDB.db, then sort it by the HighScore column in descending order. The query will then be applied to a ResultSet object. Afterward, through a while loop that runs while there are results to be found, the labels will have their text set to the values of the top three Name and HighScore rows in the database.
- checkPlayer: This method will be called by NewGamePage once a valid name has been entered in the text field and "Play!" is selected. A boolean value is declared, the query to select all values from Players where the name equals the value passed into the method, and a string to hold the name pulled from the database. After the query is executed, a loop will run to check the database for an equal value under the Names column. The string value is then set to the name, and if it is equal to the string passed to the method, it will set the boolean to true. After the loop, the boolean is returned.
- setPlayer: This method will be called from NewGamePageif the name entered in the text field was not found in the database. The query will insert the string value passed from NewGamePage into the Name column in the Players table, as well as a value of 0 in HighScore.
- checkScore: This method will be called by MainGamePage once the game ends. The string holding the name of the player will be passed to the method. An integer variable is declared at the start of the method. The query in this method will select the HighScore value from the Players table where Name equals the passed string. After finding the value, it is assigned to the integer variable, which is then returned to MainGamePage.
- updateScore: If the current high score from the database, which is found through checkScore, is lower than the score from the game, this method is called. Both the player name and current score

are passed into the method as a string and integer respectively. This method's query updates the HighScore to the passed integer in Players where Name equals the passed string. The query is executed, then the method closes.

Challenges Posed During Creation of Click It and How They Were Overcome

While developing the game, each member had come across issues that may have stalled progress on our work. One such issue that came up early on in the development of 'Click It' was how to implement a timer that was able to count down from the correct time depending on the round number as well as reset itself again and again until the game is over. After much trial and error, we realized that really all we needed was one method to control the timer that, each time it was called, would check the current screen number and round number of the game being played. Then all we needed to do is find a way to place that method in a loop where, if the round number is less than 4, each time the previous timer reaches 0, it checks what the screen and round numbers are and, if the round number is < 4, the method is called again. But here is where we ran into the real issue: The primary limitation to using the animation package's Timeline class as a recurring timer is that once Timeline.playFromStart() is called, the program does not progress past that point, and what was happening was that the countdown timer would run once, counting down from 2.5 seconds because it starts in round 1, and then the game would end. This is when we began to actually understand the Timeline class and we realized that, since the last line of the method is Timeline.playFromStart(), essentially we would have to have the entire game take place within the countdown method. What we ended up doing, is calling the countdown method initially at the bottom of MainGame's start method, but we also placed logic at the end of the method to check what the screen and round numbers are and made it so that *if the round number is less than 4, the countdown method calls itself at the bottom of the method*. Once the screen and round numbers have been accumulated such that the game is over (once round 3, screen 10 has been passed), the countdown method sees that round 4 has technically been hit, and then branches to different logic that ends the game.

Another challenge that had to be overcome during the creation of Click It was that at first, when the MainGame window was launched, it was the correct dimensions, but when the timer or button click generated a new Scene to change the button grid, the window kept moving to the bottom right of the screen. Like most issues in programming, the culprit was a simple error where we were essentially making the Scenes accumulate width and height each time a new one was launched instead of each Scene having the exact same dimensions. The fix was simple - we just changed it to where, rather than manually setting the width and height of the Scene and Stage each time a new window is shown, now, it just assigns a minimum and maximum width and height that are equal, effectively setting a default width and height without explicitly assigning it a value, thus, solving our issue and allowing each new Scene generation to appear flush and seamless to the user.

There was one issue that appeared near the end of development on DatabaseClass.java that, while minor, helped to remind us how we should always inspect code closely. After completing the getTopPlayers, checkPlayer, setPlayer, and checkScore methods, we rather swiftly added in updateScore. However, the value would not update. We tried several possible solutions to fix this. One included comparing the code to a separate version that did work, but there was seemingly no difference at the time. We also checked if we used the correct query through SQLite, and it worked correctly. After some tinkering and thoughts on what could have been causing this issue, we finally noticed a potential problem. At the end of the code that would run, prep.executeStatement used the sql string holding the query as an argument despite the try statement already having the string passed in. As this may have been causing an issue in how the string was loaded after the prepared statement was properly set, we removed this argument from the execute statement. After this simple change, the code worked correctly.