Colleen Lemak

HW3 State Management

- Document that contains:
    - **(5 Points)** The various states that an app can enter on your platform of choice
    - **(5 Points)** The various states that you must consider for your app, why you must consider it, and what must happen in each state.
- App that demonstrates:
    - **(10 Points)** Tombstone (suspend/resume) management
      (e.g. demonstrate user going away and coming back to the same state as when the user left the app without exiting the app.

Enterable App States in Kotlin Android

i. *Created*: The app's initial state. Resources are allocated, but the UI is not visible yet.
ii. *Started*: The app is now visible but not in focus. Users can see the app but cannot interact with it.
iii. *Resumed*: The app is in the foreground and "in focus", allowing user interaction.
iv. *Paused*: The app is partially obscured (i.e. by a dialog or another app); it can be resumed quickly.
v. *Stopped*: The app is no longer visible to the user but still retains memory. It can be restarted.
vi. *Destroyed*: The app is fully terminated, and all resources are released.

In addition, for Tombstone management, suspend/resume management involves **onPause**, **onStop**, and **onResume** states. Overriding these functions allow the Android app to handle going into the background and returning to the foreground with saved important data.

Various States for Numbly + Reasoning Behind Each State + Events to Happen in Each State

i. Resumed:
    a. *Why*: Users need to interact with the app and see their current progress.
    b. *Actions*: Load any saved game state or progress from previous sessions and initialize any ongoing game data.
ii. Paused:

   a. *Why*: This state indicates the user might return soon, so it's ideal for saving minimal data quickly.

   b. *Actions*: Save only essential data, such as the current guess, grid state, and any time-sensitive data, to ensure it's restored if the user quickly returns.

iii. Stopped:

   a. *Why*: The app is entirely hidden, so Android may terminate it to save resources.

   b. *Actions*: Save the complete game state, including the guessed numbers, grid layout, random number, etc., to allow full restoration if the app resumes.

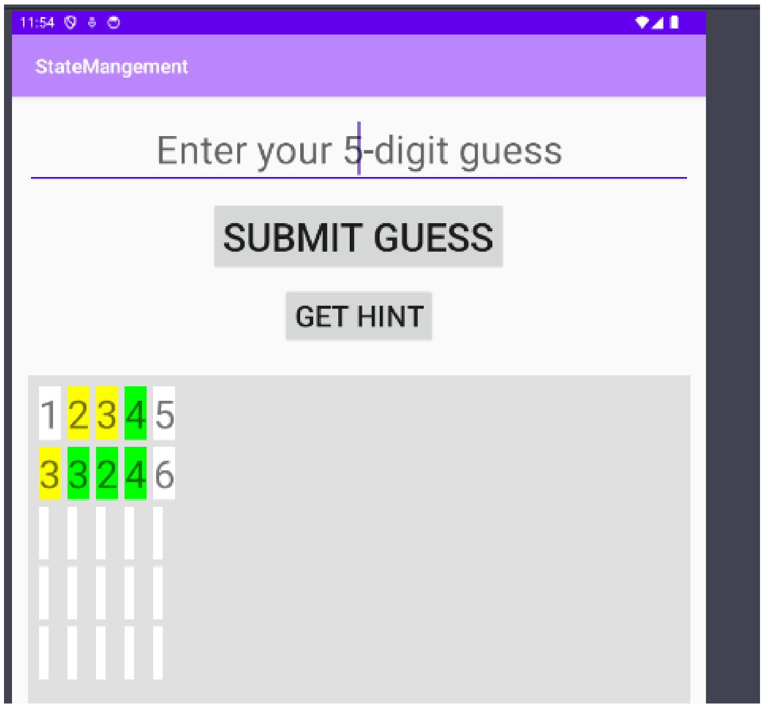
iv. Destroyed:

   a. *Why*: To ensure no memory or data is lost, the app should save all data necessary to start the game from where the user left off.

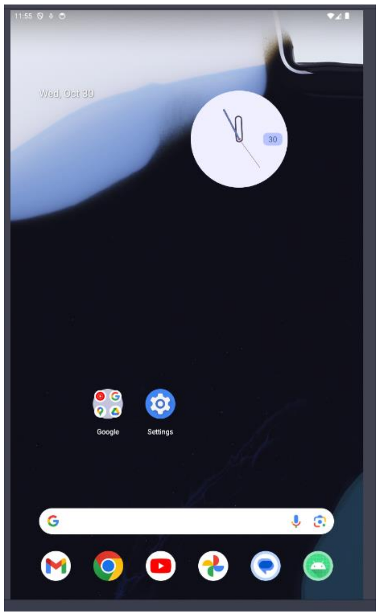
   b. *Actions*: Persist the game state, guesses, and random number in a way that allows restoration even if Android fully terminates the app.

The app will demonstrate **tombstone suspend/resume** functionality using Kotlin function overriding of **onPause(), onSaveInstanceState()**, and **onRestoreInstanceState()** functions. Additionally, I have implemented **restoreGuessGrid()**, **resetGameBoard()**, and **clearSavedGameState()** so that the grid may be reset or restored based on if the user has actually won or lost the previously active game.

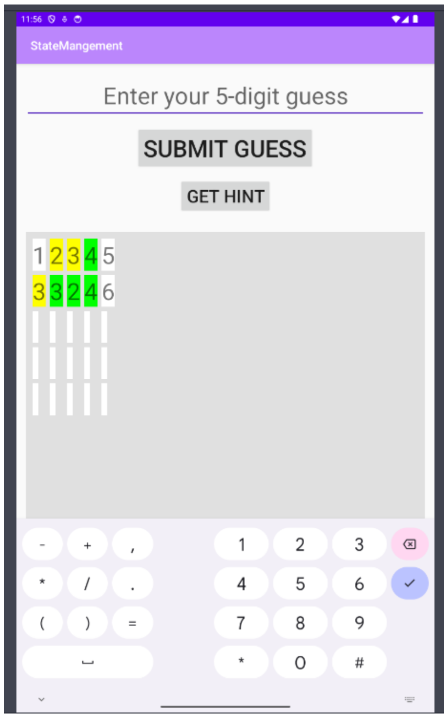
Before suspending/resuming:



Suspended/resumed by going to home screen or clicking back button:



Returning to the app:



As shown, the data is reloaded and saved if the user decides to suspend/resume the application. This is achieved by overriding the previously mentioned Android functions; by implementing **clearSavedGameState()** and **resetGameBoard()** functions, I can reset the game altogether when the user wins or runs out of guesses so that the game will not always be resumed based on if the user has won or lost the previously active game.