# BLG 354E Assignment 1 Report

*Özkan Gezmiş 150200033*

## Part1

Load an audio file named 'bayrakfm.wav' using the `librosa.load()` function. The sampling rate of the audio file is also obtained.

The song is divided into one-second parts using `np.array_split()`. This segmentation allows for processing smaller portions of the audio at a time.

For each one-second part of the song:

- A Fourier transform is applied to convert the time-domain signal into the frequency domain.
- The second half of the transformed song is extracted, leaving the first half for embedding the message.
- An empty array is created for the transformed message, with the same size as the transformed song.
- The extracted frequency components are placed at the beginning of the message array.
- An inverse Fourier transform is applied to convert the modified message back to the time domain.
- The real part of the resulting message is extracted and stored.

The individual message parts are concatenated into a single numpy array.

The amplitude of the message is normalized to ensure it fits within the range of 16-bit integers (np.int16).

The normalized message is written to a new audio file named 'message.wav' using `wavfile.write()`.

## Part2

Load the previously embedded message from an audio file named 'message.wav' using the `librosa.load()` function. The sampling rate of the message is also obtained.

The digital filter is defined by its numerator and denominator coefficients. Specifically, the system is represented by the transfer function $\mathrm{H(z)} = \frac{1-\frac{7}{4}z^{-1}-\frac{1}{2}z^{-2}}{1+\frac{1}{4}z^{-1}-\frac{1}{8}z^{-2}}$

The defined digital filter is applied to the hidden message using the `signal.lfilter()` function. This operation filters the message according to the specified transfer function.

The amplitude of the filtered message is normalized to ensure it fits within the range of 16-bit integers (np.int16). This step is necessary for preparing the message for writing to a new audio file.

The normalized filtered message is written to a new audio file named 'filtered_message.wav' using `wavfile.write()`.

# Part 3

Recording and Analyzing Audio:

- Audio is continuously recorded using the default microphone and analyzed it for peaks in the frequency domain.

- Peaks in the frequency domain signal are detected using the `scipy.signal.find_peaks()` function.

Determining Movement Commands:

- Based on the analysis of the audio peaks, the script determines the appropriate movement command to execute. Start from the move_up continue with move_right, move_left and lastly move_down. Then come back to center of the tile.

- If multiple peaks are detected, it assumes a "mined tile" scenario, where the next move avoids potential dangers.

- If no peaks are detected, it assumes a clear tile and proceeds with the next move in the sequence.

Executing Movement Commands:

- The script uses the `pyautogui` library to simulate keyboard input for movement commands.

- Single moves are performed with a short key press followed by a release after a brief delay.

- Double moves are performed with a longer key press and release, simulating a more extended action.

Handling Movement Sequences:

- The script maintains a sequence of movement commands (up, down, left, right) and iterates through them for normal moves.

- For double moves, it executes moves longer. The time is determined by me. For monster to move to the center of the neighbour tile, monster should go 1.75 seconds. But 0.4 seconds is enough to just detect the mine.

# Part 4

Two transfer functions, denoted as `s1` and `s2`, are defined using the `signal.lti()` function. These functions represent distinct LTI systems characterized by their poles, zeros, and gains.

Using the `signal.bode()` function, the frequency response (magnitude and phase) of each system (`s1` and `s2`) is computed. This response provides insights into how the systems react to different frequencies.

Separate plots are generated to illustrate the magnitude and phase responses of the individual systems (`s1` and `s2`). These plots offer a visual representation of how the systems amplify or attenuate signals and shift their phase across different frequencies.

A third transfer function, `s3`, is defined to represent the cascade (series connection) of `s1` and `s2`. This composite system represents the output of `s1` fed into `s2`.

Similar to before, the frequency response of the cascade system (`s3`) is computed and visualized through plots depicting its magnitude and phase responses. These plots help understand how the cascade of systems influences the overall output signal.

Additional plots are created to compare the magnitude and phase responses of all systems (`s1`, `s2`, and the cascade `s3`) on the same graph. This comparison aids in understanding the combined effects of the individual systems when connected in series.