TO: Professors J. Hung and V. Nelson
FROM: "The PokeSquad", Demetris Coleman and Justin Wahlers
SECTION: 003 – Tuesday 3:30 p.m.
DATE: November 18, 2016
SUBJECT: Final Project – Simple Video Game

The objective of the final project for the semester is to create a simple video game. The project requirements are as follows:

1. Have character controlled by Directional Pad (D-pad) shown on the LED Matrix.
2. Player must hear some auditory feedback when the keys are pressed
3. Must be some form of randomly generated enemy or conflict.
4. Must be a specified start and end state for the game.
5. In end state, the game should only be playable again when the RESET key is pressed.
6. Recorded score must be shown to the player at the end of the game.
7. Bonus: Design scaled difficulty into the game.

The PokeSquad chose to implement Frogger Mini as the simple video game. The design of the game was based of the video game Frogger and the project requirements. The game consists of a 1x1 LED character (frog) and various nx1 cars that move vertically along the middle 6 columns of the 8x8 LED matrix. The two end columns (0 and 7) are safe zones or "sidewalks" where the frog cannot be hit. The objective of the game is to cross the street (move from column 0 to column 7) without being hit by a passing car. To take care of the first two requirements, the 1x1 LED character is moved by the D-pad, which plays a short note depending on the key pressed. It also plays a song once you reach the opposite sidewalk.

The cars take care of requirement 3. They are randomly generated at the top of the LED matrix in columns 1 through 6. The player is prevented from moving until the first car reaches the bottom of the screen. If hit by a car the game ends and displays the score which satisfies requirements 4 and 6. While the score is being displayed, no buttons work other than reset. This satisfies the fifth requirement. If you manage to reach the opposite side sidewalk, the score is increased, the character advances to the next stage, and the speed of the cars are increased. This satisfies the bonus requirement.

Testing and Procedure

To implement the game, several states were made that allowed certain functionalities. These are described in Table 1. In lab, the first step of testing was to simply load the program and see what actually worked. Initially, there was a bit of flickering whenever a button was pressed. The cause of this was a software delay in the ISR for detecting the buttons. Shortening it by a factor of ten removed perceivable flickering.

Table 1: State function description

State	Function	Triggers
0: Initialization	Set character to start position, plays song, clears all cars	Reset button, Switch Level
1: Run	Allows player to move character and press pause	Initialization end
2: Pause	Freezes game	Pause Button
3: Switch Level	Increases score and difficulty (increase speed of cars), sends game state 0	Reach column 7
4: Game Over	Stops all functionality and sends game to state 5	Character and car collide
5: Display Score	Resets difficulty and score, displays scrolling score.	Game Over

It was also notice that the random function generator was not really random. The same pattern of outputs is produced for a given seed number. This is demonstrated in Figure 1 and 2 where two different seed numbers are sampled in multiple runs. To make the numbers random and less predictable, the variable for the seed number, SEED, was made into a global variable. Next, the SEED is assigned to a new number between 0 and 1000 generated from the same random number generator that makes the cars. This is done every time a directional button is pressed. The dependency on both time and user input make the generation of the seed value random. It follows that the random generation of the cars will be random as well. Figures 3 – 6 show the distribution and line plot of the SEED and the position of the cars for multiple runs. One flaw to this solution is that if the player never touches the buttons, the game is predictable. So the very first level is always the same when the program is first started.

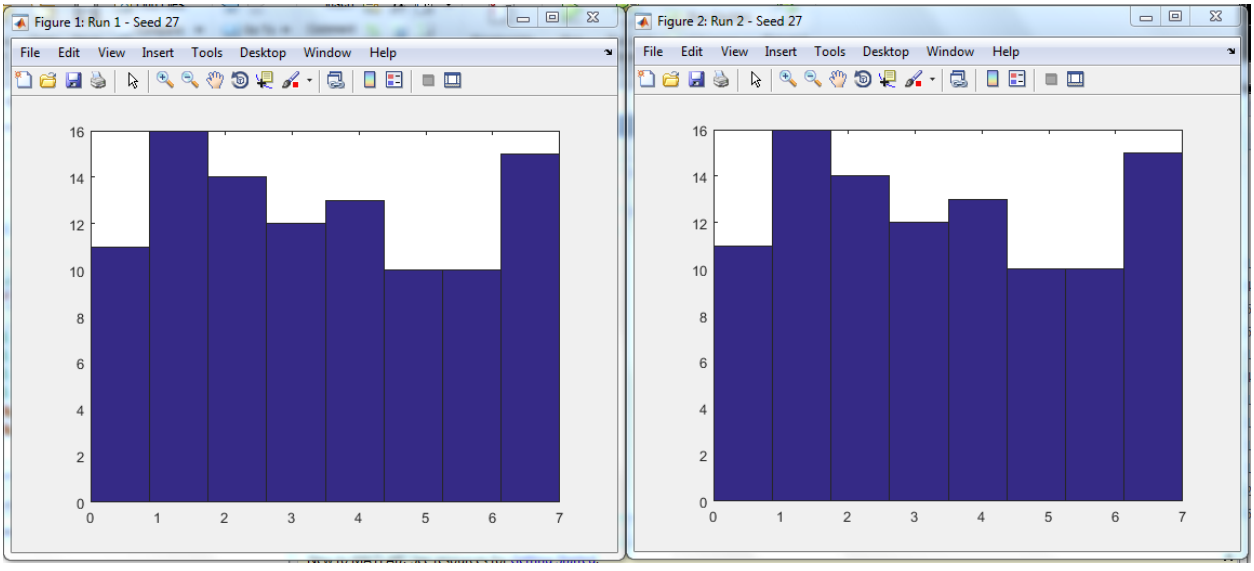

Figure 1: Distribution of first 100 generated numbers between 0 and 7 (position on LED Matrix) for seed of 27. Horizontal axis is position. Vertical axis is number of times value spawned.

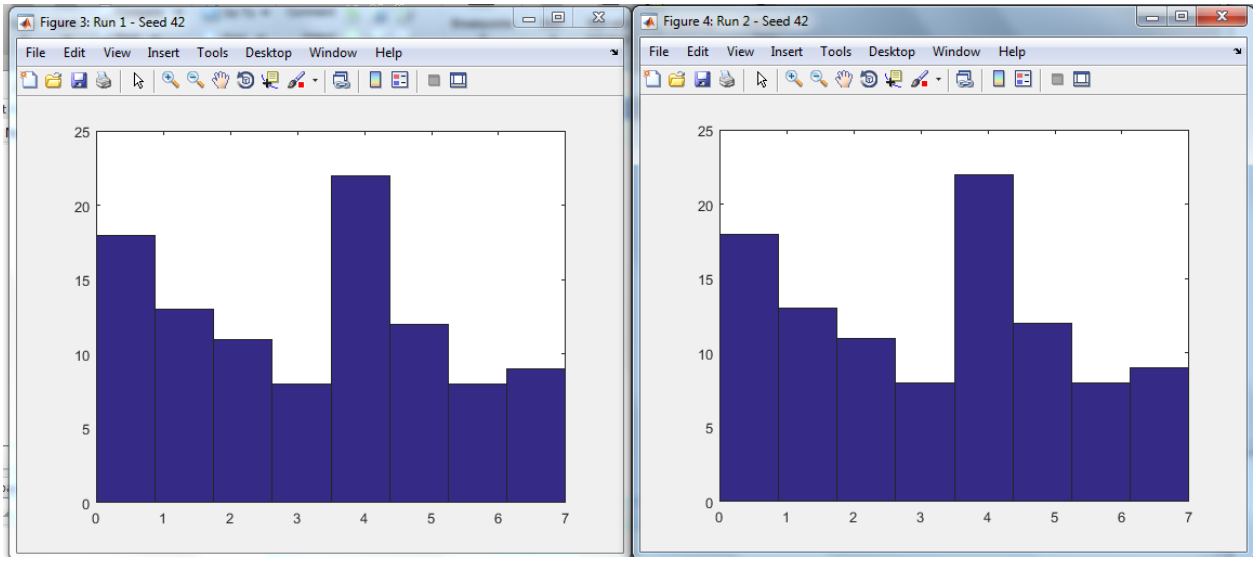

Figure 2: Distribution of first 100 generated numbers between 0 and 7 (position on LED Matrix) for seed of 42. Horizontal axis is position. Vertical axis is number of times value spawned.

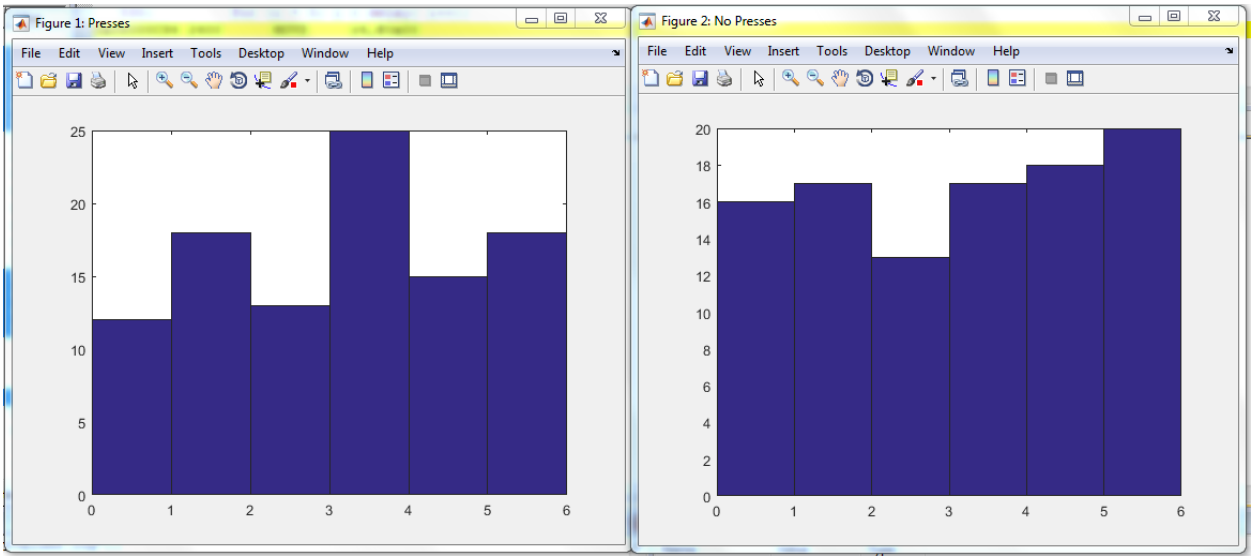

Figure 3: Distribution of first 100 generated numbers between 1 and 6 (position on LED Matrix) for varying randomly generated seeds. Horizontal axis is position. Vertical axis is number of times value spawned.

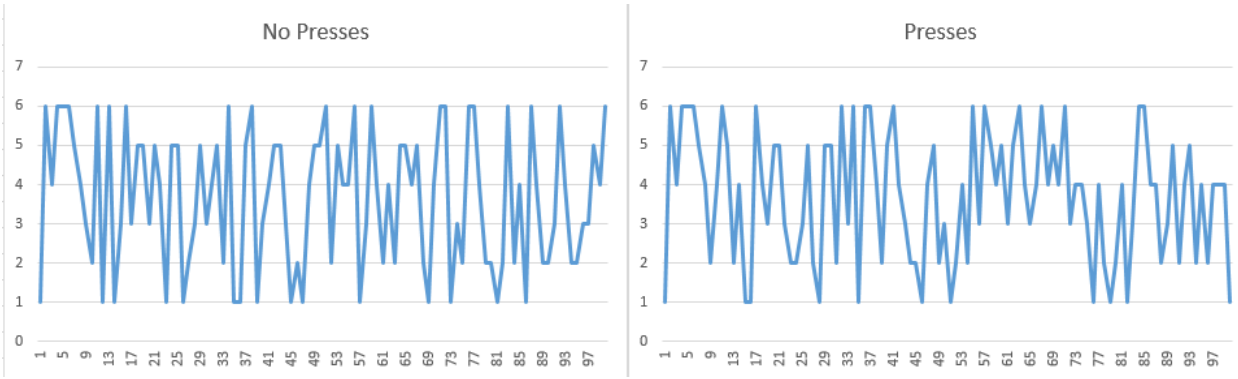

Figure 4: Plot of first 100 generated numbers between 1 and 6 (position on LED Matrix) while randomly generating seeds for two different runs. Horizontal axis is order in which position values were generated. Vertical position value.

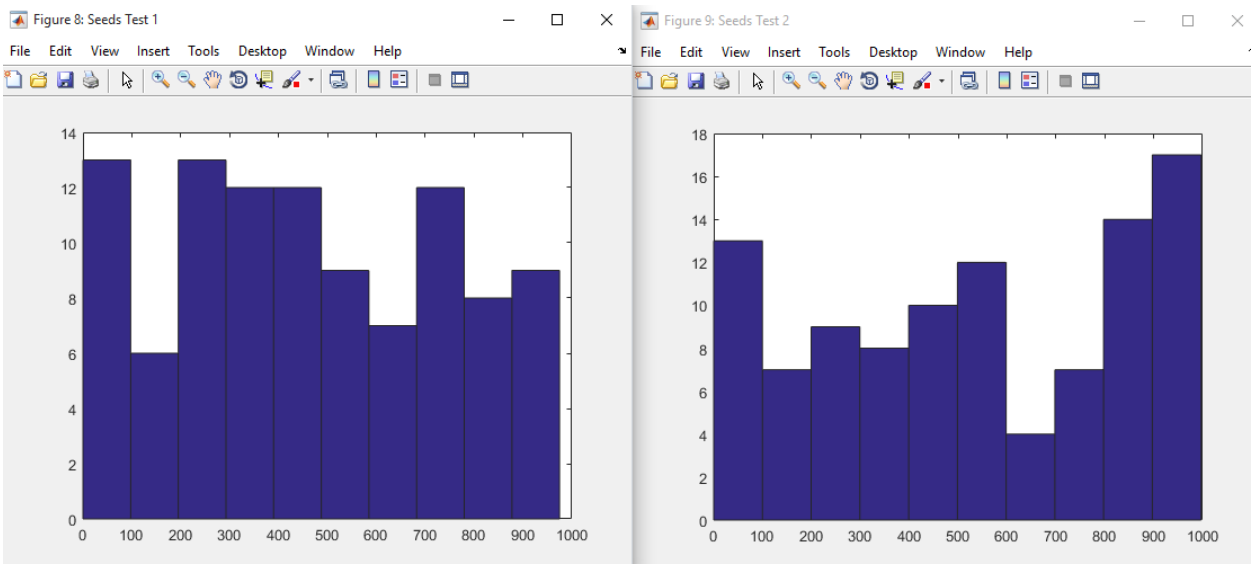

Figure 5: Distribution of first 100 Seed values for two different runs. Horizontal axis is seed value. Vertical axis is number of times spawned values fell within the range of the 10 bins (range of 100).

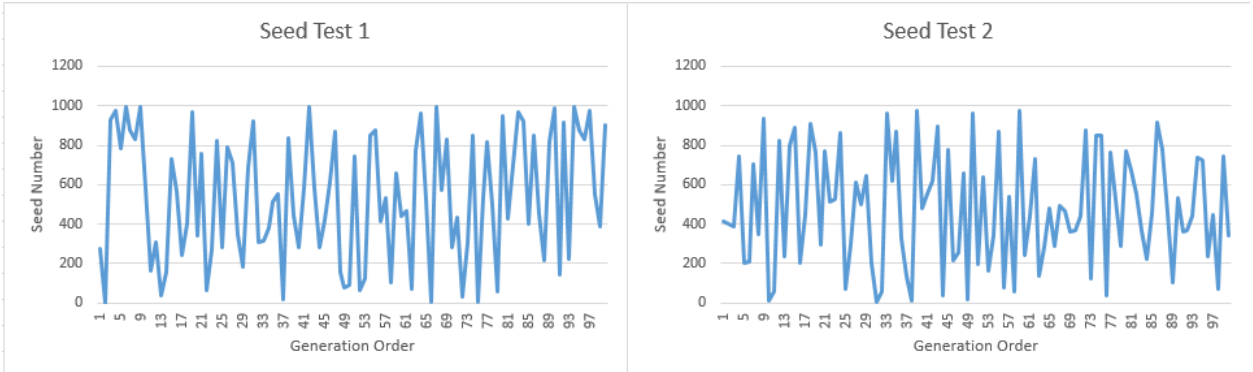

Figure 6: Plot of first 100 Seed value seed values, ordered from first to last, for two different runs. Horizontal axis is order in which values were generated. Vertical seed value.

The game was played to test that the cars crashed into the character making the game unplayable, that an increase in difficulty sped up the cars, and that the pause button worked. The debug watch window was used to watch the difficulty and the score to verify that they incremented correctly. During this testing, it was found that the player could move the character across the screen rapidly before the cars cold populate the LED matrix. This issue was fixed by extending the initialization state making the character immovable until the first car crossed the matrix.

Next, the score display was test by playing the game, watching the score variable in the watch window to verify that it matched the display. While testing this, it was realized that the difficulty and score were being reset in the initialization state and that the score display was could be interrupted by pressing the pause button. To fix this the statements to reset score and difficulty were move to the end of the function that displays the score. And the functions of the pause button were disabled if the game was in state 5.

Results

The of a 1x1 LED frog can be controlled by the pressing the directional pad which played a sound upon the character's movements. If the pause button is pressed, all game play is frozen and only reset and pause buttons are functional. Various nx1 cars that move vertically along the middle 6 columns of the 8x8 LED matrix while the two end columns (0 and 7) are left empty unless the frog is on them. If the frog is able to successfully cross the street (move from column 0 to column 7) without being hit by a passing car, the player's score is incremented according to the difficulty level, the difficulty level is increased, and the player advances to the next level. A song also plays when the frog reaches the new level. If the frog is hit, the game stops and the score is displayed as continuously scrolling text.

Conclusion

The final project was successfully implemented with all of the specifications met. It uses most of the microcontroller's functionalities, most notably the timer and interrupts, to allow a feedback loop between the player and the video game for an entertaining experience.