Owen Hushka

ECEN 3753 RTOS Final Project - Labyrinth Game - May 1, 2024

I worked for 33.83 hours out of the 12.75 I expected and I was able to complete almost the whole project.

I was able to do pulse width modulation with the LED's, but I was not able to have this correlate to the fuel left in the drone's fuel tank.

I was able to fully implement the holes, as well as I was able to detect if the board had excessive tilt in either direction. Additionally I was able to tell if the player exceeded their playing time. I was able to detect these game ending scenarios as well as display a "game over" screen which told the player what caused their game to end.

I was able to implement waypoints. I began with all of the waypoints being blue except for the first waypoint being orange. I then had the waypoints turn gray once they had been hit. I was able to detect the correct order, so the player had to go through the waypoints in a clockwise manner. If they went through all the waypoints clockwise, they won the game. If they went through them in the wrong order, they had to go back through in the right order.

I implemented values.isHardEdged, this draws a border around the maze and implements physics which do not allow the ball to leave this border.

I had the system timer to keep track of the total time elapsed and subtract this from the values.timeToComplete. I also used this to keep track of how long the button had been held down and compare this to the values.maxTime which the disruptor was allowed to run.

In the end I did not implement a menu, but I did implement a screen which says "press to play" before the game begins. I was then able to take the touch input and start the timers accordingly.

As for config data which I did not use, I did not use the waypointsReuse variable. I was not totally sure what this would be used for. Other than that, I found all configuration data to be useful and changing the values had an effect.

If the line passes through two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ then the distance of (x_0, y_0) from the line is:[4]

$$\operatorname{distance}(P_1, P_2, (x_0, y_0)) = \frac{|(x_2 - x_1)(y_0 - y_1) - (x_0 - x_1)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}.$$

https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line#:~:text=Line%20defined%20by%20an%20equation,-In%20the%20case&text=Similarly%2C%20for%20vertical%20lines%20(b,along%20a%20horizontal%20line%20segment.

For wall physics, I used the equation for the distance from a line to a point. This point being the drone's center. I used the start and ends of the wall segments as the two points for the wall. This

equation, however, I figured out was for determining the distance from an infinite line to a point. This meant that I also had to determine, for example for a horizontal line, the ball's X position - the wall's starting x position. I had this in an OR statement for the ball's X position - the wall's ending x position. I then had this OR'd for if the ball was in between the wall's starting position and the wall's ending position. I had the distance from the infinite line be less than 5 and then also the x position had to be in the right range as well.

I had some issues figuring out how to get the wall physics to work, so to help debug, I colored in the area around the wall where I was trying to implement the physics. This was fairly simple, I used the if statement for the physics, but instead had any point in this area of distance from the wall be colored in.

I was able to make the wall physics work if the drone was coming at a horizontal wall from the top/bottom, but I was not able to get it to work if it came at a horizontal wall straight from the side. The same goes with vertical walls.

I first implemented physics based on only the current angular velocity of the board. I then was able to implement physics which accounted for past velocities and then calculated the acceleration and velocity from the last 0.3 seconds and added this to the cumulation of the past velocities.

I have a mac so I was unable to get the SeggerView working to look at my tasks. I was able to use default priorities without running into any issues.

I used semaphores for inter task communication. I had the GyroTimer release a semaphore for the gyro thread to sample the gyro values. This thread then released a semaphore for the drone task to calculate the angular velocity and ultimately the drone's next position. This task accounted for holes and walls and the edges of the maze. It also checked if the button was pressed and did all of the calculations for the disruptor. This task then released a semaphore to the LCDUpdate task which then moved the drone to the appropriate position on the screen.

Next Steps:
If I had two more weeks to work on the project, I would work on implementing the generateMaze() and printMaze() functions to be able to create mazes with more cells. I do not think this would be that hard to do. I would also make it so that walls when they are hit from the wrong angle, for example vertical walls hit from the top or bottom, still cause the ball to stop.

Task:	complete/not yet complete	Estimated Time To Complete	Time Spent So far	Bold items are larger items with subtasks below them. Their estimated time to complete is a sum of the time to complete each subtask
Read and digest project document	complete	15 minutes	30 minutes	
Create Task Diagram	complete	15 minutes	2 hours	
create task list	complete	15 minutes	1 hour	It took me quite a while to really wrap my head around the project and be able to break it into smaller tasks
create config data data structure	complete	5 minutes	30 minutes	I created this structure as well as a function which loads the default values into this struct. I had some issues deciding which file to include the data structure in
create menu at start of game to allow users to set config data	**not yet complete**	**1 hour total**	**1.5 hours**	
take input from touchscreen	complete	50 minutes	1.5 hours	I did not end up implementing the menu, but I was able to take input from the screen. I had the user press on the screen to begin the game
randomly generate maze	**complete**	**3.25 hours total**	**8 hours**	
Be able to make mazes on the LCD display	complete	30 minutes	3 hours	This took much longer than expected to complete as I ran into several errors along the way. I actually thought my board was broken for a while because my LCD screen was showing a screen which seemed to be a broken screen. After playing around for quite some time with the configuration and initialization of the LCD I as able to fix the issue.
use random number generator	complete	30 minutes	30 minutes	This did not take me very long as I have quite a bit of experience using the RNG on the STM
random number for each possible wall position to determine if wall is there and if there is a hole there	complete	1 hour	1.5 hours	I was able to accomplish this without running into any major bugs. I was not sure how to use this random number, but decided that I would do the number modulo 10 and then divide the number by 10 and this was a much more managable way to use this very large random number

2d matrix of cells	complete	30 minutes	2 hour	This took much longer than expected as I was running into an error where my threads were unable to access the memory addresses that was storing the data from my pointer to the array. I was able to solve this issue by dynamically allocating memory for this array
implement waypoints	complete	30 minutes	1 hour	I implemented waypoints by making a function to draw the outline of a circle and then using the distance formula as I did with the holes
make sure no holes or walls where the first waypoint is	complete	15 minutes	15 minutes	I put the ball in the upper left corner and made it so that walls and holes did not spawn here
hard code a maze for when config maze size is 0x0	complete	30 minutes	0	
test for game ending conditions	**not yet complete**	**3 hours total**	**2.75 hours**	
be able to detect tilt of more than 90degrees	complete	30 minutes	30 minutes	
Implement hard edge	complete	1 hour	1 hour	I was able to draw borders around the maze and then stop the ball if it made it to these borders. I knew how to calculate where the borders were, so if the ball reached where the border was I just moved it back 1 pixel.
detect wheather drone is above hole	complete	30 minutes	30 minutes	I was able to use the distance formula and then if the distance was less than or equal to the radius of the whole the game ends
detect if too much time has passed	complete	30 minutes	30 minutes	I used the system timer to subtract time from value set in configuration for the time to complete the game.
calculate game score	not yet complete	30 minutes	15	I did not end up printing a score when the game was complete, but I did display the time left for the game to complete as of when the game was won
implement user botton disruptor	**complete**	**1 hour 20 minutes**	**3.33 hours**	
Implement button ISR	complete	20 minutes	1 hour	This did not take long as I was able to essentially copy what I did on previous labs. I also used a function returnPressed() which returns true if the button is pressed and false if the button is not pressed

Implement "Drone Update Mutex"	completed	20 minutes	2 hours	the inter task communication was somewhat simple
Add appropriate functionality in the drone thread	complete	40 minutes	20 minutes	This was also fairly simple. I only needed to use a couple if statements
Implement LED PWM to correlate with fullness of energy store (green) and time to charge energy store to minimum activation energy	**not yet complete**	**1 hour 45 mintues**	**2.25 hours**	
Create global variable to keep track of energy	complete	15 minutes	15 minutes	
Implement logic to correctly keep track of the drone's energy	complete	30 minutes	1 hour	
Figure out how to make the LED's PWM and calibrate it and have it reflect the energy left in the tank	not yet complete	1 hour	1 hour	I was able to make the LEDs PWM but I was not able to make it calibrate with the fuel left in the tank
Physics	**complete**	**3.25 hours total**	**12 hours**	
read gyro	complete	15 minutes	30 minutes	This was not too hard as we have used the gyro frequently in our previous labs
Approximate rotation angle	complete	1 hour	1 hour	I was able to calculate the corrected gyro velocity using the following formula: correctedGyroVelocityY = (17.5/1000) * gyroVal; //deg/sec //17.5 is sensitivity
compute drone velocity: old v + acceleration	complete	30 minutes	1 hour	I first implemented physics based on only the current angular velocity of the board. I then was able to implement physics which accounted for past velocities.
Make it so when drone hits a wall, it loses all its momentum in that direction	complete	30 minutes	7 hours	I did not think I would be able to complete this. In the end I realized that my generateMaze function was incorrectly assigning values to the start and end of the wall, which was making it so that I could not detect the distance away from the drone that the ball was

indicate to user how tipped the board is	complete	30 minutes	1.5 hours	I printed out the x and y values for velocities.
make it so when the drone hits the border of the maze, it loses its momentum in that direction	Complete	30 minutes	1 hour	This was not too difficult to accomplish. I used a bunch of "if statements"
TOTAL		12.75 hours	33.83 hours	265%