Project Definition, version 20240307.3

ECEN3753: Real Time Operating Systems

Spring, 2024

**Labyrinth** (pic from Amazon.com)

Many of you may be familiar with the tabletop marble game where you try to get the ball through a maze while avoiding traps (holes).

This semester, our project will be to utilize our STM32F429i-DISC1 board and its gyro, LCD, RNG, LEDs and button to implement an advanced version to train the next generation of ground observation drone drivers.

While the wooden version shows the expected path, our walls and holes will be randomly generated, and so our science lab has augmented our design with a quantum disruptor, which when activated by the user button has two simultaneous effects: it will allow our eye-on-the-ground to move unimpeded through a wall, or to move over a hole for a brief period of time.

We will use the 2D angle of our board “from being flat” to tip our virtual labyrinth—you could alternatively think of this as the input to your drone’s “drive motor”. The more you angle away from vertical, the more the drone will power itself in that direction. We will follow physics equations for a board tipped in a gravity field, regardless of how you like to think about it.

The drone will be lost into a trap if the center of the drone is above any part of a trap when the Disruptor is inactive.

Project Management

As you look at the rubric that is attached to the first week of the project phase of our course, you’ll see that there are some common, and some unique things required each week. We’re following an agile program management process, where there are some “Minimum Viable Product” (MVP) requirements, and many other improvements that you can pursue after the known MVPs are complete.

You won’t know initially whether certain assumptions and settings are sufficient to achieve a playable game, and that’s ok at this point! Part of the idea to Agile is to solve the most critical-path problems first, leaving degrees-of-freedom open when you can to allow shifting later when priorities change, or missing pieces become understood well enough to allow understanding of what ELSE is still required.

Ask questions (in #general) when you think that I, your “customer”, may be able to clarify requirements so that you can limit or simplify your work or if you think I inadequately described my wishes. When I can’t guarantee you reduction in complexity, you may be identifying a risk that you should show in your Risk Register (we’ll talk about that in class).

Each week, you’ll be submitting a fairly terse report on what you did during the week to get closer to the goal of a working game that meets the detailed requirements. Look to short-cut out tricky code to allow you to implement and test other parts simply, as it will speed up your development.

Part of the weekly report will be to compare how much actual effort you’re putting in compared to what you estimated in the first week. How far away from 1.0 your ratio is will NOT affect your grade—this part of the exercise is intended to inform you as a beginning engineer about how biased your estimation is. (Engineers with 20+ years of work regularly experiencea ratio of 2-4, so your likely optimism is in good company!)

Detailed Requirements

You could demonstrate a working game without many of the described details, so I encourage you to view the MVP as some combination of these detailed requirements. I’ll clarify later in the project what the points will be for each of these, but I leave it to you to consider “what makes it playable, as intended” as your guiding light.

Configuration Data

Computer systems often have a semi-static set of input data that drives our behavior. This project is no different. Levels of possible MVP delivery here might include:

1. All values are hard-coded in the source code
2. All values are #defined in a header file
3. All values are set as the default values for a compound data structure
4. All values are set as defaults as above, but are altered via a touch-screen driven menu before the game starts

As you can see, the game might be playable with only the first step above, but when on demo day we find a set of values that we want to test, your solution would prove to be less useful to the customer than even #2 would have been. #4 would “wow” your customer, but #3 would be better with a working game than to have done #4 and not have the game work.

Throughout the rest of the project description, I’ll refer to configuration data by the following structure, though you may implement them as #1 or #2 above, depending on how far you get with MVP features.

Notice that the units of the Config data is not exactly the standards you’re probably used to. This is so that, for the system we’re modeling, we can use integers for ALL of the configuration data!

If you think that you need more configuration items than what I provided, speak up in #general and explain what and why!

My current expectation of the configuration data follows. We will discuss the Config.Version in class, but for now set it to 1 and check that it is 1.

You can implement a menu system to allow the default values to be changed for a particular run, both to “wow” your customer and to make it easier for you to test varying inputs. However, keep in mind that this is less important to your customer than having physics, holes, walls, and the disruptor all working fairly well. Note that if any values are non-default for a run, that must be indicated with a game-end score (See “Game Completion”).

Config

* Version [-] : 1
* Physics
  + Gravity [kg\*cm/(s^2)] : 980
  + UpdateFrequency [Hz] : 50
  + PinAtCenter [enum:0=DRONE, MAZE] : DRONE
  + AngleGain [-/1000] : 500(PhysAngleRate = Gyro\*500/1000)
* Drone
  + Disruptor
    - MaxTime [ms] : 1000
    - Power [mW] : 10000
    - MinActivationEnergy [mJ] : 6000
  + EnergyStore
    - MaxEnergy [mJ] : 15000
    - RechargeRate [mW] : 1000
  + Diameter [mm] : 10
* Maze
  + TimeToComplete [ms] : 30000
  + CellSize [mm] : 12
  + Size
    - Width [cells] : 15
    - Height [cells] : 15
  + ObstacleProbability
    - Wall [Pr\*1000] : 100
    - Hole [Pr\*1000] : 200
  + HoleDiameter [mm] : 11
  + HardEdged [bool] : TRUE
  + Waypoints
    - Number [-] : 4
    - Diameter [mm] : 90
    - Reuse [bool] : FALSE
    - Location (x,y pairs) [mm] : (list of Number)
      * 50,50 (#0: starting point for game)
      * 130,50
      * 130,130
      * 50,130 (#Number-1 is the game’s goal, if !Reuse)

User Button: Disruptor

As long as Config.Drone.Disruptor.MinActivationEnergy is met, the user button will turn on the quantum disruptor for the shortest time of:

* Config.Drone.Disruptor.MaxTime
* Config.Drone.EnergyStore.MaxEnergy /

(Config.Drone.Disruptor.Power-Config.Drone.EnergyStore.RechargeRate)

* Time that the button is held down

The energy store is recharged at a Config.Drone.EnergyStore.RechargeRate. As you can see, there will be many data in our Config data structure—see previous section of this definition document. During the time the disruptor is active, the drone will effectively continue to travel per the board’s tipping and gravity, and upon termination the drone will quantum-tunnel instantaneously forward on its current trajectory (based on velocity) until it is no longer intersecting with any walls.

Your first MVP will likely not even implement the disruptor, but may simply note that the button was pressed (and light up an LED or print something on the screen) as you get the fundamental gyro, physics, and board display working. Thereafter you’ll want to get this working, since our maze generation does not guarantee open paths to the waypoints!

Another example of MVP and degrees-of-freedom: If you’ve barely entered a wall and the disruptor runs out of power, the drone should still tunnel forward on its current trajectory as stated above. Will I notice if you pop back to your starting point if you run out of power during tunneling? Probably. Will I care? Not as much as having your game be fundamentally playable! Also, what if there is NO position on the current trajectory that is not intersecting with any walls? That’s probably a good time for you to ask your customer whether they care, and if so ask them what they want.

For this project, playability is the MVP. Accurate physics is a very near second. Effective tunneling is not far behind because it increases playability…

LEDs: Status Indication

We have two LEDs, and getting some quick indication of system status is as useful when we’re driving a quantum-tunneling drone as when we are turning on an oven and waiting for it to heat up.

We have a green LED, which due to its color would be good to use it for something “good”. As your customer, I’m going to ask you to make that LED light at a %-brightness that is relative to the fullness of our energy store. Look into either use of the hardware PWM or doing PWM by use of OS timers to perform callbacks to turn the green LED on and off.

The red LED would be more useful to indicate “badness” (at least in the US), so your customer wishes it to flash at a rate such that each LED period will be 1/10 of the time remaining to charge the energy store to the minimum activation energy (on for ½ of that time, then off for ½ of that time). Once the period gets down to 2 clock ticks, quit doing the computation, and just schedule an on-off each pair of clock ticks until the energy store is back to minimum activation energy. Once the minimum energy for activation is available, turn off the red LED. For illustration:

Physics Modeling

While we spend a good part of our college education years using calculus on continuous-independent-variable functions, and assume that there is an analytic equation to represent all state information of our physical systems, there is little reason to do so for this project. We will be able to sample the gyroscope frequently enough to simply perform “integration” by computing [Riemann sums](https://en.wikipedia.org/wiki/Riemann_sum) (essentially “right” or “left” method, not “min” or “max”), so we can approximate the rotation angle around any axis by:

Angle[k] , for each axis, with Angle[0] set to zero.

While the rotation angles really do affect each other (see [Euler Angles](https://en.wikipedia.org/wiki/Euler_angles) and [Rotations](https://en.wikipedia.org/wiki/Rotation_matrix)), if the angles are very small, then the cross-term effects remain small (primary terms contain cos(Angle), while cross-terms contain sin(Angle)). To ensure that you get a working project, you should first make your project work with the assumption that the angles are independent to compute a new angle-of-slope at each time instant. Later, if you find this unplayable and have first-tier MVP features complete, you can implement rotation matrix corrections to make it play more realistically.

Now that you know the slope angle and direction, you can compute the drone’s new velocity as its old velocity plus the projection of gravitationally-induced acceleration in the x and y directions (a bit of trig here, to get from the slope angle and direction to the projections onto x and y axes). The ∆T that we’ll use for our updates is the reciprocal of Config.Physics.UpdateFrequency.

As your customer, let me share with you: I’m much more interested in your proper modeling of physics and taking the holes and quantum disruptor into account than worrying about maze walls initally. So for your sequence of implementation, you may want to:

1. Trust that the physics update is frequent enough that simply checking after every ∆T whether the probe’s center IS above a hole is good enough to detect this game-end condition
2. If that method misses detections, compute the distance from [every hole’s center to the line from the probe’s old and new positions](https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line#:~:text=In%20Euclidean%20geometry%2C%20the%20distance,nearest%20point%20on%20the%20line.) at every time update, and if any is less than the radius of the hole, end the game (note: a lot of that equation will not change as you go through checking ALL the holes at one time instant)
3. Keep a list of all wall end points, and check for the intersection of that line with the probe’s (oldpos,newpos) line to begin dealing with walls
4. Possibly consider 2D matrix of wall lists, so that for any approximate probe position, you only have to examine a short list of wall segments
5. Figure out what the heck you’re going to do once you realize that the probe hits a wall. Again, rely on your physics updates being frequent enough to ensure the probe’s path is properly constrained around a corner:

 

Walls have zero thickness, and collisions with walls are frictionless and inelastic. At one point you’ll perhaps be wishing that I’d let the probe also be a point-mass, but I assure you—we’re avoiding a whole lot of floating-point nastiness by NOT making that simplification. If anyone wants to convince me that non-zero wall thickness is easier, I’m open to your sales pitch…

Another place where there are MVP degrees-of-freedom: the Labyrinth game is pinned at the center of the maze. That is, the z value of the center of the maze does not change. If the platform is rotated very fast when the ball is near the edge of the maze, you can get the ball to launch into the air a bit, and with our ability to change gravity or make the board huge in our simulation, this effect could be VERY significant. While modeling the true game might make you want to set Config.Physics.PinAtCenter=MAZE, =DRONE is much easier to simulate correctly and eliminates one of the possible exception conditions that we are supposed to notice in Tilt Indications, below. (You can even thoroughly justify it being BETTER if you are viewing your board as the driving input system for a drone, rather than the Labyrinth game.) The only requirement about MAZE and DRONE is that if the configuration data indicates something your code doesn’t support, you halt and indicate why, instead of launching the game.

Note that there are 2 ways to make your system less sensitive—and they will act differently. You can reduce the physics gravity, or reduce your AngleGain. When AngleGain is 1000, a 90 degree rotation of your DISC1 board will cause your simulated maze to be in a vertical plane. When AngleGain is 150, a 90 degree rotation of your board would cause the maze to tip at 150\*90/1000 = 13.5 degrees. Note that AngleGain >1000 can cause issues… either test your inputs to ensure it’s in the range you expect and support, or handle those strange edge cases.

Tilt Indications

The double-meaning here is that we care about both feedback to the player/driver to indicate the Angle of tilt of the board and the tilt direction, and to provide “game over” conditions when the game is lost (e.g. to the TILT condition of a pinball game). Your customer needs you to indicate if the ball leaves the maze, due to any of:

1. Hop (perpendicular to the board), due to (Config.Physics.PinAtCenter=MAZE and sufficient torque to launch the ball), or some cases if AngleGain>1000
2. Maze departure (in the plane of the board), due to (Config.Maze.HardEdged=FALSE and you don’t tip the board adequately to keep the center of the ball from going past the maze edge)
3. The physical board (not the maze’s plane) exceeds 90 degree tilt from flat.

Your customer also advises that they really think it will be a good idea to have some indication on the board of “how tipped” and “which direction tipped”, but is willing to have you think about how you would like to implement this. (You may have an MVP idea that simply prints text, and another for better playability once the MVP work is complete that shows a “downhill” arrow on the display whose length indicates the magnitude of the tipping, or you might light up 1-2 edges of the board where the brightness of that edge indicates the steepness in that direction, for example.)

Maze Generation: Cells, Walls, Holes, and Waypoints

In addition to using real units (such as mm) for some of our dimensions, we will also define our maze by both “cells” that may have walls on the sides of, or holes in them, and by the paths that we are expected to follow in traversing the maze. After all, the goal of the game is to make it through a maze that I’ve simplified by approximately bounding the resolution to “cells” on the board. Place the (x,y) origin at the upper-left of the board (which avoids negative positions during successful play, by construction) which will ensure that my definition of waypoints look the same as yours.

You are to create a random maze at the start of the game, determining where interior walls are placed and where holes appear. (Exterior wall existence is determined by Config.Maze.HardEdged.)

Here are a couple examples of how 4 waypoints with diameter K could be configured with their X and Y spacing, S, of and , respectively. Note that a current position in an intersecting area between 2 or more waypoints meets the conditions for all of the intersecting waypoints at once. This can lead to some interesting conditions that you might need to detect… like if the drone is in the center of the left picture, how many waypoints are traversed in one system update?



For each of the potential wall positions that have not been considered yet during maze creation, generate a random number (using our TRNG hardware for full credit, but perhaps PRNG for MVP) to determine whether there should be a wall there. Use similar logic (with the respective, different Probability value from Config, applied once per cell) to determine if there is a hole in the center of that cell.

Note that before you can move or activate the quantum disruptor at the start of the game, your game could be prematurely terminated if the drone is above a hole. Therefore, if a hole is generated within the Config.Maze.HoleDiameter of the first waypoint, that hole would result in an unplayable game. You can either inhibit holes in cells that are too close to the first waypoint, or throw away a maze that results in such and generate a new maze—your choice. A similar but utterly invalid situation would arise if the random maze generated a wall whose nearest point to the first waypoint was less than Config.Drone.Diameter/2. (MVP for both conditions: just make the user hit the reset button again!)

Your customer definitely would like the next waypoint that is needed to be drawn as a circle of some type on the maze. Beyond that, you can try showing more waypoints to assist playability as long as it is clear which one is NEXT.

An optional MVP maze option is that if the Config.Maze.Size is 0 by 0, you can use a hardcoded maze defined by you. This can be very useful if you want to put off gyro, LCD, and button handling, because you can still do significant unit testing to check your physics implementation. Again, if you don’t accept randomized maze parameters or those that are provided exceed your support, your program must indicate the failure to the user rather than launching an improperly-configured game.

Game Completion

When the game completes, indicate completion and a win-or-lose indication, as well as a “score”.

Loss:

* Probe went out of bounds in the maze-plane (not possible for Config.Maze.HardEdged=TRUE)
* Probe went off the maze-plane (only possible if Config.Physics.PinAtCenter=FALSE or Config.Physics.AngleGain>1000)
* Too much time went by (more than Config.Maze.TimeToComplete) without making it to:
  + - at least 1 waypoint (Config.Maze.Waypoints.Reuse=TRUE, noting that waypoint#0 does not count here), or
    - all waypoints (Config.Maze.Waypoints.Reuse=FALSE)

Win:

* Probe made it to all waypoints within the time limit (Config.Maze.Waypoints.Reuse=FALSE), or
* Probe made it to at least one non-initial waypoint (Config.Maze.Waypoints.Reuse=TRUE)

A game score is:

* Waypoints/second for (Config.Maze.Waypoints.Reuse=TRUE), and the total time for all waypoints for (Config.Maze.Waypoints.Reuse=FALSE).
* Do not show a score for a Loss.

If you implemented a means to override the default Configuration and any default values were changed for the run, indicate so with the score.