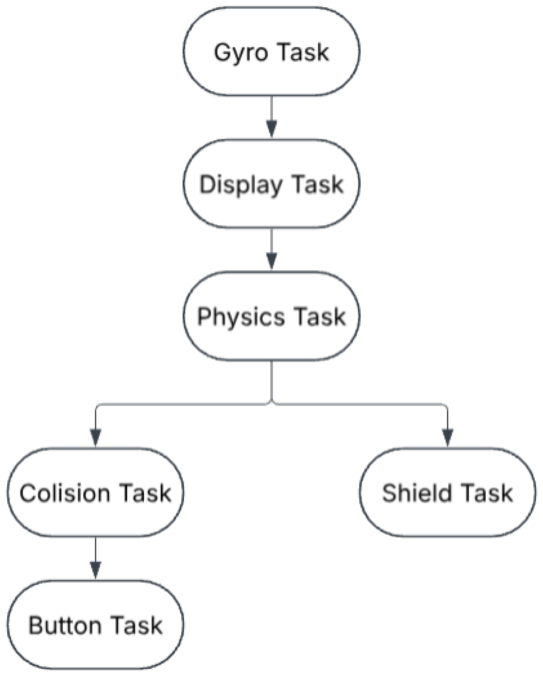
Joel J. Ramos Silva

Final Lab Write Up

Task Diagram:



The RTOS project is implemented with the following tasks, each responsible for a core gameplay or hardware feature. Tasks interact via shared global state, event flags, and direct variable access for simplicity and determinism.

Task Name	Priority	Description
DisplayTask	Below Normal	Renders game elements (platform, cannon, projectiles, satchels, castle, energy bars, Game Over screen). Uses LCD driver and font utilities. Reads platform_x to place the cannon.
PhysicsTask	Normal	Updates satchel/projectile positions, applies motion physics, handles spawn/despawn logic based on game state and difficulty.
CollisionTask	Below Normal	Checks for all collisions: satchels hitting shield or platform, projectiles hitting cliffs, castle, or player. Triggers Game Over/Win when needed.
ButtonTask	Low	Detects press-and-hold (railgun charging) and double-tap (shield activation). Sets event flags for shield task or railgun fire.
ShieldTask	Low	Manages shield energy drain, visual bar update, deactivation upon depletion.
GyroTask	Low	Reads gyro Y-axis velocity every 50ms and stores filtered value for platform movement.
DefaultTask	Normal	Placeholder — not used in final game loop.

ITC (Internal Task Communication):

- GyroTask : updates velocity value read by DisplayTask.
- ButtonTask : raises flags or sets state for ShieldTask and Railgun_Update().
- PhysicsTask : updates projectile/satchel positions used by DisplayTask and
- CollisionTask : updates game state (Game_Win(), Game_Lose()).
- Shared variables: platform_x, platform_velocity, shield_active, projectile.active, etc.
- Mutexes not required due to update-order control and design simplicity.

Effort vs. Estimate Tracking

At the start of the project, I estimated approximately 36 hours would be required to complete all major components of the game. I tracked the time spent on each part of the project and compared it to my original estimates. Here's a breakdown of the effort versus actual time spent:

- Game concept design and task diagram
  Estimated: 2 hrs | Actual: 2 hrs
  I spent time defining the gameplay, key tasks, and building the initial task diagram to guide the architecture.

- LCD display setup (LTDC, fonts, energy bars)
  Estimated: 4 hrs | Actual: 5 hrs
  More time was required than expected to properly draw and clear platform elements, handle text rendering, and display energy bars with live updates.

- Gyro input and platform movement
  Estimated: 3 hrs | Actual: 4 hrs
  I had to tune deadzones, apply smoothing, and manage position constraints, which took slightly more time.

- Button press and double-tap detection
  Estimated: 3 hrs | Actual: 3 hrs
  This part went as planned. I implemented both short press for the railgun and double-tap for shield activation.

- Railgun mechanics (charging and firing)
  Estimated: 3 hrs | Actual: 4 hrs
  Handling energy scaling, drawing the charge bar, and implementing projectile logic took additional iteration.

- Shield system (activation, energy drain, bar display)
  Estimated: 3 hrs | Actual: 4 hrs
  I implemented energy draining over time, bar updates, and satchel deflection, which required more tuning than expected.

- Satchel spawning and physics
  Estimated: 4 hrs | Actual: 4 hrs
  This aligned well with the estimate — I randomized velocity, applied gravity, and managed spawn limits by difficulty.

- Collision detection (satchels, shield, projectile, platform)
  Estimated: 4 hrs | Actual: 5 hrs
  Detecting all edge cases (e.g., self-hit, shield drain, castle/cliff hits) and triggering state changes took more debugging and testing.
- Difficulty system and menu
  Estimated: 2 hrs | Actual: 2 hrs
  I used the gyro to navigate and select difficulty levels, and adjusted parameters like spawn rate and energy drain.
- Game win/lose behavior and display
  Estimated: 2 hrs | Actual: 3 hrs
  I added a Game Over screen, blinking LED animation for prisoner escape, and edge case handling for repeated shots.
- Wall crash logic and late-game polish
  Estimated: 3 hrs | Actual: 3 hrs
  Implementing loss by fast platform-wall collision, tuning thresholds, and testing different crash behaviors matched expectations.
- Final testing, bug fixing, and documentation
  Estimated: 3 hrs | Actual: 2 hrs
  Final testing and cleanup went a bit faster than expected since the gameplay loop was stable.

Total Estimated Time: 36 hours
Total Actual Time Spent: 41 hours

Overall, I was surprised on how close I stayed to the original plan. A few areas like collision detection, graphics, and shield mechanics took longer than expected, but others were more efficient. The additional time was well spent ensuring the game was fully functional, smooth, and polished for demo and turn in day.

Risk Register Summary:

At the beginning of the project, I identified several technical and design risks that could impact the success of the game. Throughout development, I updated the risk register based on what actually happened, how I mitigated those risks, and whether they affected the outcome.

Risk	Initial Concern	Mitigation Strategy	Outcome
Button press detection	Risk of unreliable press or bounce causing misfires or missed inputs	Implemented debounce logic and used press duration to distinguish between single and double taps	Resolved — both press types are stable
Gyro noise / over-sensitivity	Jerky or unstable platform movement due to raw gyro values	Added deadzone and moving average filter to smooth input	Resolved — movement is smooth and responsive
Display not initializing	Screen might stay blank due to LTDC misconfig or timing issues	Ensured correct initialization order and HAL delay after I2C init	Resolved — display consistently works
Satchel collision not registering	Risk of satchels passing through shield/platform without detection	Tuned bounding boxes, added separate shield/platform checks	Resolved — collisions behave correctly
Wall crash loss logic	Platform might bounce when it should crash (or vice versa)	Refactored logic: platform only bounces if below velocity threshold	Not Resolved — Does not end the game at high speed
Game Over not resetting cleanly	Risk that projectiles or shield remain active after Game Over	Reset game state and cleared active elements on state change	Resolved — Game Over screen and logic are clean
Win condition edge case	Player might shoot castle during escape and cause bug	Added check: second castle shot during escape = instant loss	Resolved — logic is enforced
Resource constraints	Running out of CPU or memory due to tasks or rendering	Used static allocation, optimized task priorities and screen updates	No issues encountered

All the key risks I initially identified were either fully resolved or avoided by early testing and incremental development. In a few cases, unexpected behavior (like the wall collision logic) required last-minute tuning, but none of the risks caused major delays or forced design changes. Careful use of debugging prints and structured task separation helped keep everything under control.

Functional Test Results

I performed functional testing throughout development and again after implementing all core features. Each gameplay mechanic was tested independently and then in full-game integration to ensure proper behavior. Below is a summary of the functional tests and outcomes.

Feature-by-Feature Test Results:

- Platform Movement (Gyro):
  Tested across full x-axis range. Smooth response to gyro input with proper deadzone and boundary behavior.
  Pass
- Railgun Charging & Firing:
  Press-and-hold behavior charges energy; releasing fires a slug based on charge level. Verified vertical projectile, charge bar updates, and no-fire if energy is 0.
  Pass
- Shield Activation & Drain:
  Double tap activates shield; drains capacitor energy over time. Shield blocks satchels and has its own health bar. Shield deactivates on energy depletion.
  Pass
- Satchel Behavior:
  Satchels spawn at intervals based on difficulty level. They fall with gravity, bounce on walls, and deactivate on shield or platform hit.
  Pass
- Collision Detection:
  All major collision types tested:
  - Satchel → Platform → Game Over
  - Satchel → Shield → shield health reduced
  - Slug → Platform → Game Over if unshielded
  - Slug → Castle → starts prisoner escape
  - Slug → Cliff → tracks hit count
    All collision logic performs as expected
- Wall Crash (Platform):
  Fast impact with wall causes Game Over; slow impact causes bounce. Value from the Gyro was not working well so the function could not be fully applied
  Fail
- Difficulty System:
  Gyro-based menu allows user to select Easy, Medium, or Hard. Each mode affects

satchel spawn rate, shield drain rate, and platform forgiveness.
Pass

- Win Condition – Prisoner Escape:
  Shooting the castle once and the cliff twice triggers escape. LED blinks for 5 seconds; if castle is shot again during this time, game is lost.
  Pass
- Game Over Screen:
  Displayed correctly on all loss conditions. Game can be restarted by holding the button.
  Pass
- Menu Navigation:
  User can select difficulty from menu using gyro tilt and confirm with button.
  Pass

Conclusion:

All core game features were tested and passed. I confirmed full functionality through real-time gameplay on hardware, with each subsystem performing reliably. The final build is stable, polished, and demonstrates all required mechanics.