**Functional Tests**

The following functional tests were performed during the in-class demonstration of my project.

1. Once the platform attains a velocity which exceeds the maximum bounce velocity and strikes a vertical wall, it is destroyed, ending the game.
2. The cannon fires using parabolic paths according to its current charge level to hit the foundation or wall of the castle. Upon the full depletion of either zone’s health, the evacuation is initiated. Upon the completion of the evacuation, the game ends.
3. When triggered and possessing the minimum energy level, the cannon shield neutralizes any satchels which come within its radius.

**Project Status**

My project performed exactly as expected during the in-class demonstration. All requirements for the project were accomplished and the game was fully playable. All game parameters were able to be easily changed allowing for gameplay adjustment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Time Estimate** | **Actual Time Used** | **Status** |
| Task Diagram | 3 hours | 5 hours | Complete |
| Physics hand examples | 3 hours | 3 hours | Complete |
| Write Input Tasks | 5.5 hours | 7.5 hours | Complete |
| Write Physics Tasks | 8 hours | 8.5 hours | Complete |
| Write Display Tasks | 8 hours | 5.5 hours | Complete |
| Integrate Tasks | 3 hours | 2 hours | Complete |
| Debug Final Code | 5 hours | 4 hours | Complete |

Table 1: Project Progress

**Solutions Analysis**

The vast majority of my project was completed within app.c. It has a length of 1390 lines. While the supporting files such as gpio.c and queue.c were also essential, they did not compose the bulk of my algorithm and code writing as they were already written in previous projects within this course.

My Physics task was updated along the same schedule and period (100 ms) as my other tasks (excluding my LedOutputTask which had a period of one second). This approach was not conducive to adjusting the code or altering the scheduling in any way. If one wished to implement another method of scheduling and triggering than round robin and periodic, large portions of each task would need re-writing. Despite these issues, my project still functioned well and executed its tasks on time as required. All objects were updated every time the Physics task ran. This allowed for easy real-time comparisons of each objects current position and accurate detections of collisions.

My simulation parameters all required a relatively strict balance for the game to be playable. This was largely due to the slow refresh rate that each task was restricted to by the OSTimers. This meant that it the parameters were out of balance, objects would appear to randomly fly around the screen with no reasoning behind why they had move to those new positions. Many times the game would end almost immediately as the objects had intersected with each other very quickly and depleted any tracking counters.

If I had two more weeks to work on this project, I would have implemented more of the extra features such as multiple satchels being in flight at a time and having a startup menu. Also I would have implemented solutions for higher refresh rates on the LCD screen. This would have enormously improved the playability of my project.