Initially, I tried a purely random approach. The algorithm would randomly place all five components on the board and then check if the layout was valid. As in output in the log file, this failed thousands of times. The three components on the edges pull the board's centre of mass significantly to the sides. For the layout to be valid, the remaining two components (MICROCONTROLLER and CRYSTAL) must be placed in a very specific, small area to perfectly counterbalance the other three. A purely random placement is just the hit and trial approach that will almost never land them in that exact spot. So therefore, after carefully examining all the constraints I implemented a **"Constraint-Driven"** approach.

The algorithm successfully showed the desired results that was much more intelligent because it is **constraint-driven**. Instead of placing components randomly and hoping for the best, it uses the constraints to guide its decisions.

* The algorithm starts by placing the most constrained components first—the three that must be on an edge. It randomly picks an orientation for the two MIKROBUS connectors (either on the left/right or top/bottom edges) and a random edge for the USB\_CONNECTOR.
* **Calculated the "Balance Point"**- After placing the three edge components, the algorithm calculates their combined centre of mass.
* It knows the final centre of mass for all five components must be very close to the board's centre (25, 25).
* Using this information, it calculates a **target coordinate** where the centre of the remaining two components (MICROCONTROLLER and CRYSTAL) needs to be to achieve perfect balance.
* **Targeted Search, Not Random**- Instead of searching the entire board for a place to put the MICROCONTROLLER, the algorithm now searches only in a small area around that calculated **"balance point"**. This dramatically increases the chances of satisfying the Global Balance constraint.
* The approach is guided by logic, it doesn't need thousands of tries. It typically finds a fully valid solution that satisfies all seven hard constraints within just a few dozen attempts, making it extremely fast and reliable.
* **It is Non-Deterministic,** if we run the solver.py script again, it will produce a new and different, yet still perfectly valid solution. This proves its dynamic and robust nature, which is the core requirement of the assignment.
* This combination of random exploration, rule-based logic, and a physics-based heuristic makes the algorithm both efficient and effective, allowing it to solve a complex spatial puzzle in under a second.

PLOT OUPUT