SDL2 Collisions Performance Report

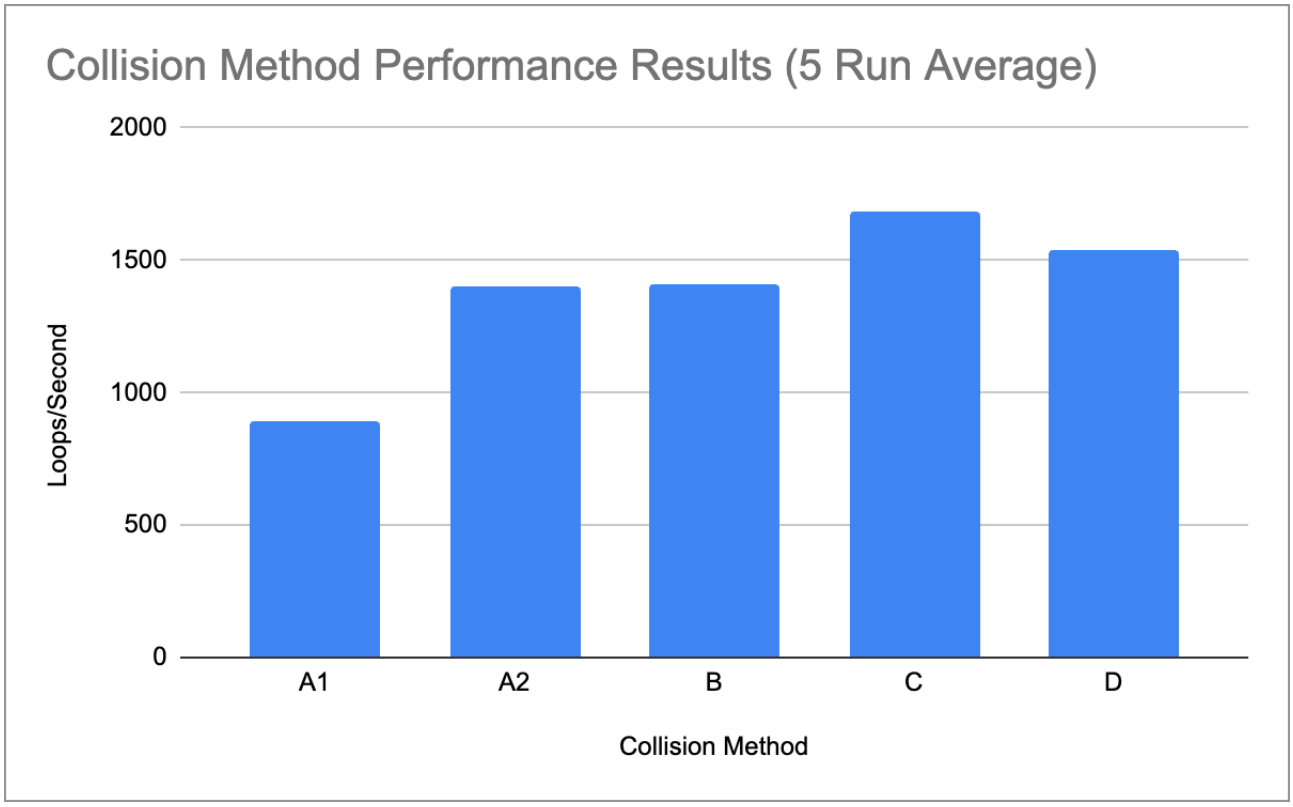
Hayden Whiteford, 104001272

I. Initial Performance Review

The provided code provides 5 initial collision methods:

- A: uses nested for loops to iterate through a said box, and all of the boxes that it could collide with. The collision function itself uses the index numbers as arguments.
- A2: once again using nested for loops, but this time keeping the nested for loop to stay one index ahead so a box never checks collision with itself.
- B: using A2's loop system to stay one index ahead, but this time the collision function itself now takes in copies of the boxes themselves rather than just index numbers.
- C: virtually the same as B, but this time the arguments have the & symbol, meaning direct references to the boxes themselves are used, instead of making copies of the boxes.
- D: a slight variance on method C, in which the collision code is cut down quite a bit due to the function not using any temporary local variables.

All of these methods are quite similar to each other and each only have slight changes to each other, but the underlying collision logic and iteration is the same for all. In my testing I determined method C to be the fastest, although many times C was ofter overtaken by A2 or D.



Test	Run 1	Run 2	Run 3	Run 4	Run 5	Average
A1	933.33	883.33	871.38	908.36	878.04	894.888
A2	2272.67	1161.61	1192.67	1201.27	1178.55	1401.354
B	2275.67	1161.28	1180.61	1201.27	1211.93	1406.152
C	3743.67	1165.67	1169.28	1153.28	1189.27	1684.234
D	3085.67	1164.95	1118	1152.57	1180.27	1540.292

II. Further Performance Increases

There were multiple avenues I explored in increasing the performance further. The first method I explored was spatial partitioning, in which we split the screen into “cells”, meaning that a box would only check for collision of other boxes in the same cell. The logic required to code this cost just as much as the gain I was getting, and performance was lacklustre. Similar tweaks to the logic, such as using a less accurate collision method like using a radius, or measuring the distance to eliminate far away boxes still cost too much.

The final solution was to simply add in the Break; line inside the for loop.

```
void crash_test_all_E()
{
    // check i against j
    for (int i = 0; i < BOX_COUNT; i++) {
        for (int j = i+1; j < BOX_COUNT; j++) {
            if (crash_test_D(boxes[i],boxes[j])) {
                boxes[i].state = CONTACT_YES;
                boxes[j].state = CONTACT_YES;
                break;
            }
        }
    }
}
```

For the new Crash Test E, it simply uses the same collision function as D, but this time it includes a the break line as soon as the collision is returned True.

I also disabled the rendering, and increased the time of the test by a second.

With the new results, we can see the new Method E is now significantly faster than all other methods.