**Collision Detection and Depth**

When we look at collision detection methods, it's important to understand the weaknesses of using Axis-Aligned Bounding Boxes (AABB). AABBs are the quickest algorithms to use when checking if two objects have overlapped with each other, or collided. This is ideal when the objects are not rotating as it can be checked with a simple logical comparison, however the problem arises when dealing with rotating sprites, especially in frameworks such as SFML. Rotated objected require extra trigonometry equations which can be slower to calculate than with a non-rotated object, due to the needing to be checked every frame, the dimensions of the bounding box are changed to continually wrap around the object (MozDevNet, *3D collision detection - game development: MDN*)

Another problem that may arise is that whilst AABB may not be too taxing on resources, they can become inefficient when we need to test every possible pair of objects in real-time. Techniques like space partitioning can help reduce the number of tests, but since AABBs are constantly moving, we need to create many indices to keep up. This can lead to performance problems, memory issues, and even program crashes, especially in larger games.

To overcome these limitations, we can consider two alternative collision methods: Bounding Circles and Oriented Bounding Boxes (OBBs). Bounding Circles have their own pros and cons. When used correctly, they can improve upon AABB collisions. Bounding Circles don't require rotation calculations as they’re invariant, so they are not affected by frame-by-frame rotation. (MozDevNet, *3D collision detection - game development: MDN*)

They are also relatively easy and efficient to calculate. However, some shapes, like flat ground or platforms, don't fit well into circles. Additionally, even though the bounds don't need to rotate, the sprites themselves may need to rotate, causing accuracy problems as shapes change at different angles. (Jeff LanderBloggerFebruary 03, *When Two hearts collide: Axis-aligned bounding boxes* 2000)

OBBS are also rotationally invariant and do no not need to be recalculated every frame; instead, they only adjust their corners. However, compared to AABBs and Bounding Circles, the calculations involved in OBBs are more computationally expensive.

(*Rigid body collision detection - scss.tcd.ie*)

Due to the evidence that I have gathered above, I am going to use a combination of AABB and Bounding Circles. Neither the character nor the tiles will have any need to rotate, so AABB will be ideal this situation. For the grenades, these will be bouncing around the screen as well as rotating during gameplay, and their circular shape can fit the bounding circle perfectly.

After carefully considering these collision methods, I have decided to use AABBs for the player and tile platforms since they don't rotate and don't fit well into circles. For the grenade and grenade launcher objects, I will use Bounding Circles as they can be represented effectively by circles and may rotate during gameplay. I could use OBB, but for the size of game and type of mechanics it will feature, but it really isn’t worth the performance trade off when I can achieve adequate results using the other two methods.

**Sources**

Jeff LanderBloggerFebruary 03, 2000 (2000) *When Two hearts collide: Axis-aligned bounding boxes*, *Game Developer*. Available at: <https://www.gamedeveloper.com/programming/when-two-hearts-collide-axis-aligned-bounding-boxes> (Accessed: 30th May 2023).

MozDevNet (no date) *3D collision detection - game development: MDN*, *Game development | MDN*. Available at: <https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection> (Accessed: 30th May 2023).

(No date) *Rigid body collision detection - scss.tcd.ie*. Available at: <https://www.scss.tcd.ie/John.Dingliana/cs7057/cs7057-2011-08-BVH2.pdf> (Accessed: 26 June 2023).