Testing

Unit Testing

Unit testing seeks to take a small, manageable element of code and provide it data to process, asserting that the result from that processed data is what was expected. In a typical environment, Unit testing is automated, to ensure that a developer does not need to manually test individual aspects of their code every time a change is made to a pathway that uses that section of code. This is a highly effective and efficient way of testing code, and frequently squashes bugs resulting from incorrect inputs, incorrect outputs, or incorrect processing.

Identified Issues

In order to conduct unit testing, test inputs have to be created in the manner that data is expected to be received by the function. Writing an identical function that takes in a differently formatted data could be seen as an approach to combat having to write incredibly complex data, however, you are then testing a unique piece of code, as it has to handle the test data differently. As a result, this is a flawed approach. If constructing valid test data in an identical format is an issue, it becomes impossible to unit test. That is unfortunately largely the case with my game thus far.

Current Approach

To effectively test functionality along a similar idea to unit testing, I can create a series of testing maps/environments within the game, allowing me to engage each element of a script in an orderly way. I must avoid attempting to test several features at once, as I may struggle to narrow down the culprit lines within my scripts. For every piece of functionality I write, I will need to write tests for all interactions between that function and other game objects or functions. This will be particularly important, I foresee, for the intended item system.

Tests:

1 Player

1.1 Player Motion/Rotation

- Should be able to move horizontally, vertically, and diagonally at approximately the same speed.
- Should be able to rotate infinite times in either direction.

1.2 Player Collision

- Player collision should work the whole way round the perimeter of the model, bar the cannon, which has no collision.

2 Enemy

2.1 Enemy Motion/Rotation

- Should be able to move freely around the map, at a constant speed, tracking and approaching the player even through objects.
- Should be able to rotate, if the enemy requires rotation of a cannon.

2.2 Enemy Collision

- Enemies should be able to collide the whole way round its perimeter colliding with the player, walls and other enemies alike.

3 Damage

3.1 Projectile Movement/Rotation

- Projectile should always shoot out of the cannon in a straight line directly, at the same angle as the cannon

3.2 Projectile Collision

- Projectile should destroy itself on collision, and should not collide with other bullets

3.3 Projectile Damage

- Projectile imposes a set amount of damage it inherits from the enemies damage stat

3.4 Projectile Rate of Fire

- Projectiles are shot at a set rate, not instantly

3.5 Chaser Collision

- Chaser can collider with all surfaces
- Chaser deals damage to players on collision

3.6 Chaser Damage

- Chaser deals correct damage, only on collision

3.7 Chaser Rate of Damage

- Damage is done only a given number of times a second, on collision