P1 - Physics: Lost In Space

200007413

1 Overview and Design

This project implements a simple arcade game inspired by the classic arcade game Asteroids. Through the creation of a simple physics engine tracking position, velocity, acceleration, rotation, and torque, the game simulates the movement of a spaceship in space represented by a simple triangle geometry.

2 Technical Description

This game is inspired by the classic arcade game Asteroids, integrating some more complex physics mechanics than the original. The aim of the game is to destroy asteroids which appear from the edges of the screen with a constant velocity. As the game progresses, more asteroids spawn at any one time, and their velocity becomes faster.

The player is a triangular spaceship which shoots projectiles in the direction faced. The player controls enable shooting, thrusting forwards and backwards, and rotation.

When a large asteroid is destroyed, it is split into a certain number of smaller asteroids, as if the projectile has broken the asteroid into pieces. Points are scored for each asteroid destroyed, with increasing points applied when smaller asteroids are destroyed.

Using inspiration from the platformer game Hollow Knight, damage to the player is the same regardless of what causes it. The player begins with 3 ”engines”, allowing damage 3 times to the player before the game ends. When damage is done to the player through collision with a projectile or asteroid, one of the engines is destroyed. After taking damage, a short period of invincibility prevents the player from losing their engines too quickly if trapped between asteroids. This is implemented as a ”shield” feature.

2.1 Physics Objects

All objects in the game that are affected by physics inherit from the `PhysicsObject` class. This class gives each object a position, velocity, acceleration, rotation, and torque. The `PhysicsObject` class ensures that each child class has a `draw()` method to render the object to the screen.

Physics objects must also define mass and inertia values as inverse values. This makes

force calculations easier, and are used to calculate positional and rotational acceleration.

2.2 Force Registry

The `ForceRegistry` class is used to apply forces to objects. This class stores a list of objects and the forces that are applied to them. `ForceGenerators` allow forces to be created and applied to objects. The primary example used is to apply user input to the player object. This `UserInput` class enables keyboard input to enact forces onto the player for thrusting and rotation.

2.2.1 Rotation

As mentioned, the `UserInput` class handles applying rotation to the player object. In the original Asteroids game, rotation is applied directly to the player, meaning that rotational momentum is not conserved. This game implements angular momentum, meaning that the player's rotation is affected by torque applied when pressing the rotation keys. This means that the player will continue rotating after the key is released, and the player must overcome their previous rotation to stop spinning, adding a layer of complexity and realism to the game.

In the original Asteroids game, momentum is maintained, and there is no drag on the player's movement, as is the case in space. This means that objects will continue moving forever if no additional forces are applied. However, rotation is done through a direct change in the angle of the player. This game implements angular momentum in rotation, meaning that when the player presses the rotation keys, a rotation force is applied to the current rotation speed.

2.3 Collisions

In each draw cycle, the main loop checks collisions between all the physics objects in the game. As different collisions require different responses, collisions are checked in sections based on the types of objects colliding.

However, the `CollisionCheck` class is used to check whether two objects have collided depending on the object shape. Subclasses of the `PhysicsObject` class are created for each shape to implement the general collision logic. The projectiles, asteroids, and power-ups are represented as circles, and the player is represented as a triangle. The `CollisionCheck` class enables collision checking for each of the combinations of shape types.

For circles and circles, the distance between the center points of the two circles is calculated. If the distance is less than the sum of the radii of the two circles, then the

circles are colliding.

For a circle and triangle, three checks are made to determine if the circle is within the triangle; if the triangle is within the circle, or if the circle intersects any of the triangles edges. If any of these checks are true, then we know the circle and triangle are colliding.

2.3.1 Bounces

When physics objects collide with each other, the overall momentum of the system is conserved. This is achieved through calculating the impulse of a collision, and applying this impulse to the objects involved. The impulse is calculated by finding the relative velocity of the objects, and the collision normal. The impulse is then applied to the objects based on their mass and velocity, making objects with less mass bounce with a higher velocity than objects with more mass.

The player is given a high mass to ensure that the player does not bounce off of asteroids with too much velocity. This is to improve the player's ability to regain control of the ship after a collision.

2.4 Projectiles

The player can shoot projectiles in the direction the player is facing. Spawned at the head of the ship, the projectile is given a velocity in the direction the player is facing. An equal and opposite force is applied to the player to simulate the recoil of the projectile. The projectile is destroyed when it collides with an asteroid or leaves the screen.

2.5 Power-Ups

Power-ups allow the player to regain engines (health), or enable the player's shield for a short period. Power ups have a small chance of being spawned when an asteroid is destroyed. Power ups are spawned with a random velocity. They are collected by the player colliding their ship with them.

2.6 Asteroids

Asteroids are spawned at the edges of the screen with a random velocity. When an asteroid is destroyed, it is split into either two or three smaller asteroids, totaling the size of the original asteroid. A minimum asteroid size is set, and when an asteroid whose size is less than three times the minimum size is destroyed, the asteroid is guaranteed to split into two smaller asteroids. When the asteroid is smaller than two times the minimum size, it is destroyed completely.

The mass of an asteroid is directly proportional to it's size, meaning smaller asteroids exert less force on objects they collide with. This makes smaller asteroids likely to become faster than larger asteroids through collisions, where larger asteroids will impart more force on smaller asteroids, causing the smaller asteroid to move faster, and the larger asteroid to move slower.

2.7 High Scores

A high scores system is implemented through the `HighScores` class, which reads in, and writes to, a `highscores.csv` file. Once the player is destroyed, the player can enter three alphanumeric characters as a player identifier. If better, the score then replaces the lowest scoring entry.

3 Conclusion and Critical Review

Overall, this project successfully implemented a physics engine with a simple physics engine enabling forces to be applied to objects of different basic shapes. A robust collision system was implemented, allowing for customizable collision responses based on the objects colliding. The game is fun to play, and the addition of torque and angular momentum to the player's movement adds a fun challenge to the game.