Deadly Dodgeball Documentation

Kieran Gara

January 21, 2024

1 Introduction

Deadly Dodgeball is my submission for the Ubisoft Next 2024 Programming Challenge. It is a 2D projectile physics based player vs. environment game, in which the player has to defeat increasingly difficult waves of robot attackers. The player has a fixed amount of health and must survive all 5 stages to win the game by dodging the enemies' projectile attacks and hitting the enemies with their own projectiles.

2 Technical Features

2.1 Two types of projectile physics

The game uses two different methods for determining the necessary projectile motion.

1. For the player's projectiles the position of the mouse relative to the player is used to determine the initial slope at which the projectile is fired. This forces the player to think about how they aim using their mouse to hit a desired target as opposed to just clicking on a target to get the projectile to go there.

2. The motion of the robots' projectiles are determined by looking at the position of the target (the player) relative to the robot and using this to determine the initial slope of the projectile needed to hit that target. This requires a more complex equation (since the y displacement is not necessarily 0) and has multiple possible solutions. The solution with a lower time of flight and trajectory was chosen as it makes the game harder for the player and avoids ceiling collisions.

2.2 Collision monitoring

The game performs collision monitoring to see when the different types of projectiles hit the different entities. Collision monitoring is also used to prevent the player from leaving their designated area with the arena of play

3 Class Breakdown

This section will give a high level description of the functionality of each of the classes. For a more in depth breakdown of the individual functions within each class please see the in-code documentation.

3.1 Player Class

The Player Class dictates the flow of the entire game, interfacing directly with the application initialisation, Update, Render and Shutdown functions. It uses all of the other functions to initialize and display the projectile and enemy objects. It also tracks the current state of the game in terms of player health and enemies remaining to determine when to move on to the next stage and when the game is over.

3.2 Enemies Class

The Enemies Class controls the position and active status of the robots. The class utilizes a randomization function to cause the robots to have constant random movement. The class also provides functionality for setting an Enemies object's status to active or not. This is utilized by the Projectiles Class for checking if an Enemies object is "alive" and can be hit or not.

3.3 Projectiles Class

The projectiles class controls the shooting and moving of projectiles by the player. The class implements functionality used to calculate the angle of the mouse relative to the player's sprite when they shoot and uses this as the initial angle of the projectile. It then uses this information to calculate the projectiles motion using the initial speed, initial angle and the acceleration due to gravity. The class also contains functionality for determining when the projectile goes out of bounds or hits an enemy.

3.4 EnemyProjectiles Class

The EnemyProjectiles Class extends the Projectiles Class, thereby inheriting its functionality. However it overwrites some of these functions for EnemyProjectiles objects as the shooting functionality is different for enemy projectiles than player projectiles. To shoot, it determines the position of the player's sprite relative to the enemy sprite shooting the object and finds the smallest launch angle possible for the projectile to hit that spot. From there the motion is the same, but instead of checking whether an enemy is hit functionality is used to check whether the player is hit and reduce they're health accordingly.