Lecture Eleven: Dodgeball (Part I)

Dodgeball Sample Usage and Controls

Dodgeball begins by displaying a splash screen. From here it enters the UI system. The UI cursor is moved using the mouse or left analog stick, and UI buttons (Start, Quit, etc.) can be selected using the left mouse button or Gamepad RDOWN button.

When the game is in play the aim is to move the character so he does not get hit by the balls. The following table lists the control while the game is playing.

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>Game Pad</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>W/w</td>
<td>Left Analog</td>
<td>Move character toward camera direction</td>
</tr>
<tr>
<td>S/s</td>
<td>Left Analog</td>
<td>Move character opposite the camera direction</td>
</tr>
<tr>
<td>A/a</td>
<td>Left Analog</td>
<td>Move character left</td>
</tr>
<tr>
<td>D/d</td>
<td>Left Analog</td>
<td>Move character right</td>
</tr>
<tr>
<td>Mouse &amp; Left Mouse Button</td>
<td>Right Analog</td>
<td>Orbit the camera around the main character within defined constraints.</td>
</tr>
<tr>
<td>Mouse Wheel</td>
<td>Right and Left Triggers</td>
<td>Move camera closer to character (roll wheel forward) and further away from the character (roll wheel backwards).</td>
</tr>
<tr>
<td>ESC</td>
<td>Start &amp; Select Button</td>
<td>Quit Demo</td>
</tr>
</tbody>
</table>

A game finishes after 20 balls have been fired, and control returns to the UI system. The game also ends if Slim is shot out of the playground.

Automatic character control will activate if the demo runs for 10 seconds without controller input. The main character will then follow a random pattern. Any control input will cancel this mode and return control to the player.
Homework Eleven

Maybe combine terrain demo sample with the kid and implement scaled-down version of game.

Dodgeball Code Overview

In this section we review the important classes that make up the Dodgeball sample.

The DodgeBall Class

The top-level application class is derived from Gamebryo’s HSample class. It is primarily responsible for three areas of functionality:

- Initialization: The Dodgeball class initializes the basic application and all of the other classes. It creates the custom UI system and initializes the PhysX SDK object. Initialization is done in this class because it is the central starting point for the application.
- UI and Game State Management: The Dodgeball class manages the current game and UI state, such as which splash screen is up or whether the game is in play. It also manages the custom UI created for the splash screens and handles input. We put UI functionality in this class because it has top-level control of the application and contains the access to inherited HSample functions.
- Update: The basic per-frame update code is contained in the Dodgeball class, although virtually all the work of update is inherited from the HSample class or done by the PlaygroundManager. Update is done here because this class is derived from the HSample class and can inherit functionality.

UI Manager Classes

Dodgeball implements a custom UI built on top of HUIManager classes. The custom UI is designed to display the splash screens and buttons (the CustomUIGroup and CustomButton classes). It is also responsible for the camera navigation system (the HSampleCustomController class) and hence the interpretation of all user input.

Playground Manager

The PlaygroundManager class performs most of the work in dealing with the game animation, and it manages the character and cannons. Its functionality can be broken into 3 areas:

- Playground management: The PlaygroundManager class is responsible for loading the playground assets and providing information about them, such as the playable area and the locations of cannon mount points.
- PhysX Scene management: The PlaygroundManager object creates and owns the PhysX3Scene object and is hence the primary contact point for the PhysX scene. In a step that comes post initialization (when all PhysX content has been created) it sets up a contact reporter so that the application knows about collisions. This information is restricted to collisions between the ball and the character using the actor group information.
- Update: The update functionality is broken into two pieces in order to facilitate asynchronous physics simulation and to enforce the requirement that no input be given to the physics system while it is being simulated. The pre-update update fetches simulation results that were begun in the previous frame and pushes any results to the Gamebryo scene graph. Note the use of the previous frame time as the time for which results are requested. User input and processing is done now, while simulation is known to be stopped. The post-update function updates all the other objects with the current time before re-starting simulation.

Cannon and Ball Management

The cannons and balls are managed by the CannonManager class and the Cannon class. The latter deals with targeting the cannon – setting its pose to follow the character while it is targeting to fire. Its update function returns true when the cannon is in a position to fire.

The CannonManager is responsible for loading the asset for both the cannon and the ball. The cannons are cloned to create one for each mount point. Twenty ball clones are created at initialization so that they are available