CMPT766 - Computer Animation

Programming Assignment 3:

Rigid Body Simulation (15 points)

Due date: **23:59 Tuesday Nov 12, 2024**

In this assignment, you are going to implement a rigid body simulator using forward dynamics. You will need to simulate the motion of a dice when it is dropped from mid-air and collides onto the ground. The coding environment will be in Unity.

Unity Setup

To begin, download and extract **CMPT766-Assignment3.zip**. Open the extracted folder **CMPT766-Assignment3** as a Unity project in Unity Hub and select **2022.3.46f1** as the editor version. Later versions should also work, but if you ran into issues then use the listed version. It should take a while to open the project for the first time.

Once the project is opened, open the scene in **Assets** → **Scenes** → **Assignment3** to begin the assignment.

Project Script files

All the C# script files are contained in **Assets/Scripts** folder. Here are some brief details on what they do:

1) **CameraController.cs**

   This script controls the camera movement. If you like, you may modify this script to have the controls bind to different keys.

2) **StateController.cs** Do not edit this file.

   This script adds key controls to control the simulation state.

3) **Matrix3x3.cs** Do not edit this file.

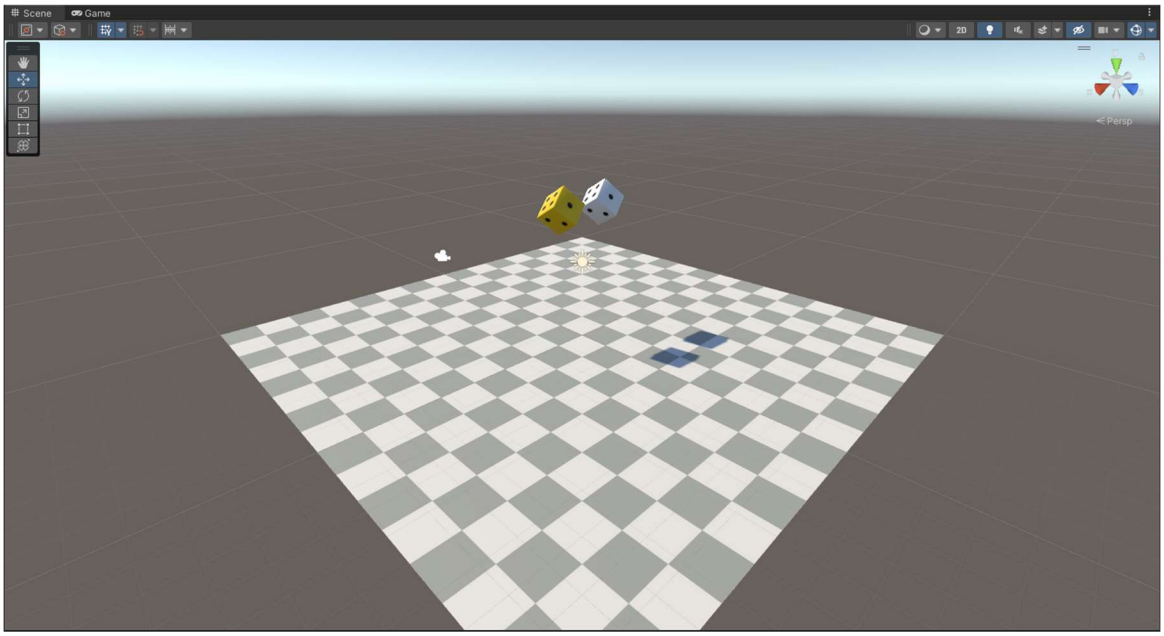
   This script defines a 3x3 Matrix class which you can use for this assignment. The documentation is documented in the file.

4) **DiceRigidbody.cs** The only file you need to modify.

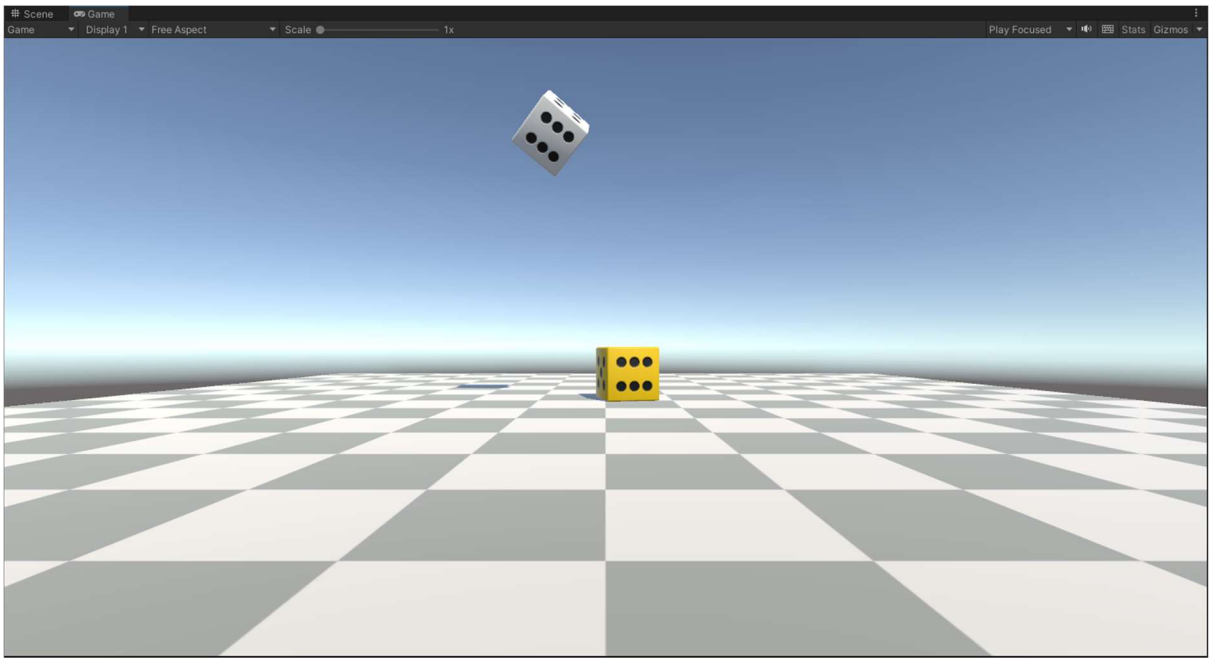
   This script will be the file that simulates the motion of the dice. All the forward dynamics of the dice should be implemented here.

Introduction to the project

When you first open the project, you'll be greeted with two dice in the scene, a white die and a yellow die.

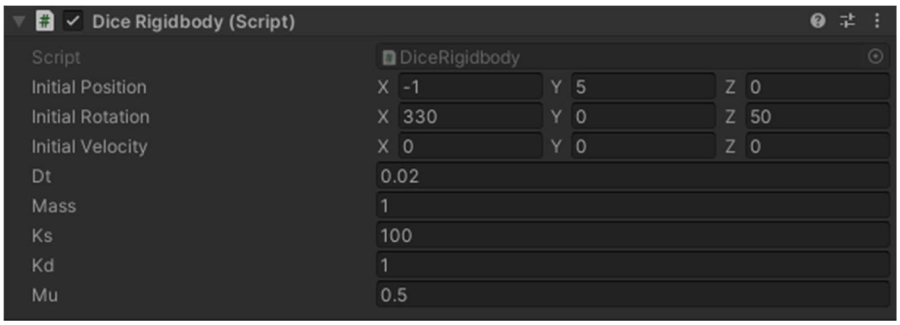


The white die will move according to your implementation of the forward dynamics. The yellow die follows the dynamics from Unity's built in physics engine. When you first run the project, the white die will not move as nothing have been implemented yet. However, the yellow die will begin to fall and collide with the ground.



You can press ‘r’ to reset the simulation state, press ‘space’ to pause the simulation, and press ‘right arrow’ to advance a single timestep while the simulation is paused.

There are some simulation parameters you can change via the Unity editor interface. Click on the **WhiteDie** GameObject and you’ll see some properties you can modify under the DiceRigidbody script component. Here are the default values when you first open the project:



Here is a brief description on what they do:

- **Initial Position:** The initial position of the white die. This also changes the initial position of the yellow die, and it will be 2 units away from the white die on the *x*-axis.
- **Initial Rotation:** The initial rotation of both the white and yellow die.
- **Initial Velocity:** The initial velocity of both the white and yellow die.
- **Dt:** The simulation timestep of both the white and yellow die.
- **Mass:** The mass of the white die.
- **Ks:** The penalty coefficient.
- **Kd:** The damping coefficient.
- **Mu:** The coefficient of friction.

Note that the parameters **mass**, **ks**, **kd**, **mu** are exclusive only to the white die and does not affect the yellow die.

Part 1: Collision Detection

(2/15 points) Complete the following function:

```
private List<Vector3> GetCollidedVertices()
```

This function should return a list of die’s vertices which have collided onto the ground. You can assume that the ground is at height zero, thus a vertex is in collision when its *y* value is negative.

You are given a list localVertices which contains the local positions of the die’s vertices that are relative to its center of mass. You must first transform them into global positions before checking for ground collision.

Part 2: Calculate Forces and Torques

(5/15 points) Complete the following function:

private (Vector3, Vector3) ComputeForceAndTorque()

This function should return the net force and net torque of the current simulation state. You can assume the simulation has a gravity of -9.81. The penalty method should be used to compute the collision force on each collided vertex, and you will need to take account of the damping force and dynamic frictional force. You must aggregate each vertex's collision force and torque into the net force and torque.

You will need to use the properties mass, ks, kd, mu to compute the relevant forces. Their values can be modified via the editor, as described before.

Part 3: Integrate Timestep

(5/15 points) Complete the following function:

private void Integrate(float deltaTime)

This function should update the position, rotation, linear_velocity, and angular_velocity state variables with timestep deltaTime. It should be implemented using the Semi-Implicit method.

You can get the inertia matrix of the die via GetInertiaRefMatrix() function. Note that this returns the inertia matrix in the die's local body frame, you'll have to transform it into the global frame given the die's current rotation state.

After you've implemented part 1 through 3, you should see the die starts to fall and tumbles on the ground. If you've implemented everything correctly, your results should be similar to the result shown in **demo1.mp4** in the demo folder. The default parameters are used in this demo.

Part 4: Parameters Tuning

(1.5/15 points) As you may see, the simulation of the white die with the default parameters does not resemble the simulation of the yellow die. Try to play around with the parameters and report the best values you can find that can result in a similar motion as the yellow die's in a new text file **answer.txt**.

For reference, you may have a look at **demo2.mp4** to see what the simulation can look like with some better parameter values.

(1.5/15 points) In **answer.txt**, write a few sentences to explain how you can achieve different motions by tuning each parameter.

Submission

Please create a zip named as **studentid.zip** (e.g. **123456789.zip**) containing your **DiceRigidbody.cs** and **answer.txt.** At the top of your files, please leave a comment that includes both your **name** and **student id**.

Submit your zip file on CourSys.