Milestone 2 – Physics Simulation

General:

- **This is an *__INDIVIDUAL__* assignment.**
- **Use Unity version: <SEE SYLLABUS FOR REQUIRED VERSION NUMBER>**
- **Late Policy:** See Canvas Course site for late policy.

Description

In this assignment, you will be creating a number of physics objects and explore some programmatic control of the physics simulation. You should use the project you submitted for M1 as the foundation. If you aren't happy with your submitted M1, you can use the original github project: https://github.gatech.edu/IMTC/CS4455_M1_Support

Tasks (100 points total)

Make sure the HUD displays your full name.

Implement the following objects in view of the camera at the start of your game (in 16:9/16:10 aspect). Read everything first to have an idea where to put the different objects.

While a few material colors are provided for you, you will need to create some of them! Be aware that the color of objects for tasks is very important, and the color will be assessed as part of grading.

Basic Rigidbodies with Collision Events

Make three (3) **blue** rigidbody spheres aligned roughly vertically that topple and roll off one another as soon as the game starts. You may need to slightly offset horizontally to avoid perfectly balancing. Modify the blue spheres to generate an event (Assets/Scripts/AppEvents/BombBounceEvent already provided) upon collision (e.g. OnCollisionEnter()). This event will be consumed by the AudioEventManager and the bombBounceAudio (glass clink sound) will be played. The crates that are already in the scene have a CrateCollisionReporter script that you might find of interest for completing this task. You can for instance, implement a BallCollisionReporter script that is nearly the same.
(5 points)

Physics Layers Example
Make three (3) **red** rigidbody spheres aligned roughly vertically that belong to a physics layer that does **not** allow collisions with other members of the same layer. Assigning Layers and tweaks to the Physics Layers Collision Matrix are necessary. The end result should be the spheres collapse into and interpenetrate with one another when the game starts.
(5 points)

Compound Collider for Complex Geometry
Using the Asset Store, search for "Free Japanese Mask" and import it (looks like a stone mask of a fox and the model is named "Kitsune"). Place it in your scene somewhere and scale size uniformly by 10.0. Next add a rigidbody controller to the Kitsune GameObject. Add a few child GameObjects that each contain a simple collider like capsules, cylinders, and boxes but are otherwise invisible. Make sure these children do **not** have rigidbody components attached. The

union of these colliders should approximate the visual dimensions of the mask and collectively provide a good collision shape. This means that the mask should be able to be moved around the arena by the player, and visual elements of the mask should not clip into the ground or other objects, nor should the mask have portions that float unrealistically in the air.

Orient the mask such that it falls over and tips on the nose upon start of the game (example: https://mediaspace.gatech.edu/media/VGD+-+M2+Mask+Demo/1_8zv0rdf0). This is an example of a compound collider physics object. The goal of falling over and the noses hitting the ground is to demonstrate that you have designed compound colliders that reasonably represent the visual geometry of the mask. It's ok if the mask doesn't settle with the nose still touching the ground so long as we see the nose interact with the ground. No visual part of the mask should substantially pass through the ground. (DO NOT USE A MESH COLLIDER FOR THIS TASK.)
(10 points)

Chain with Joint Constraints
Create a yellow chain out of at least five (5) rigidbody GameObjects of your choosing and form the chain with ConfigurableJoints. Hang it from up in the air somewhere. Orient the chain such that it is swinging slightly at the start of the game.
(10 points)

Mecanim-Animated Kinematic Elevator
Add an empty GameObject and name it ElevatorRoot. Create a **blue** child cube GameObject called Elevator with zeroed out transform position and rotation. Scale it to look more like an elevator platform. Add to the Elevator a rigidbody component configured to be kinematic. With the Elevator object selected in the Hierarchy, open the Animation Window. Click the "Create" button within the Animation Window. Save the animation as "Elevator" under Assets/Animations. Confirm that the Elevator GameObject now has an Animator component with AnimatorController of same name assigned.

On the Animation Window, add a property of Transform->Position (click the plus to add it). Add a keyframe midway through your animation so that the Elevator rises and then goes back to the original position. Confirm the animation is set to loop (Inspector view of the Elevator animation selected in the Project View Assets). Test it out. You may need to adjust the speed multiplier in the Animator window view to slow things down. If you don't like the placement of the elevator you can move the "ElevatorRoot" without redoing the animation. You get similar benefits if you want to make a prefab of the elevator and clone it multiple times.

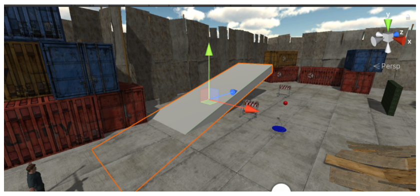
Set the elevator's Animator component update mode to "animate physics".

Now add a **red** rigidbody sphere on top of the elevator that gets pushed up and then falls repeatedly as the elevator goes up and down.

If you experience tunneling, double check the "animate physics" setting. Also try making elevator and ball larger and slow down the elevator. Lastly, you can try changing the Rigidbody Collision Detection mode settings.
(10 points)

Customized Center of Mass
Create a **green** Weeble Wobble or Punching Bozo. These are rounded bottom-heavy toys that cannot be knocked over. Make it out of a capsule collider and a custom center of mass via the Project:Assets/Scripts/Utility/RigidbodyCenterOfMass MonoBehavior with accompanying RigidbodyCenterofMassHandler editor extension. (Be sure and read the script source to understand how it works!) Place it at an angle so it is rocking back and forth at the start of the game.
(10 points)

Default Friction
Create a purple rigidbody box and place it on the ramp named BetterRamp in the Hierarchy. You can see that it is gray and against a wall. Confirm that it does not slide down the ramp.
(5 points)

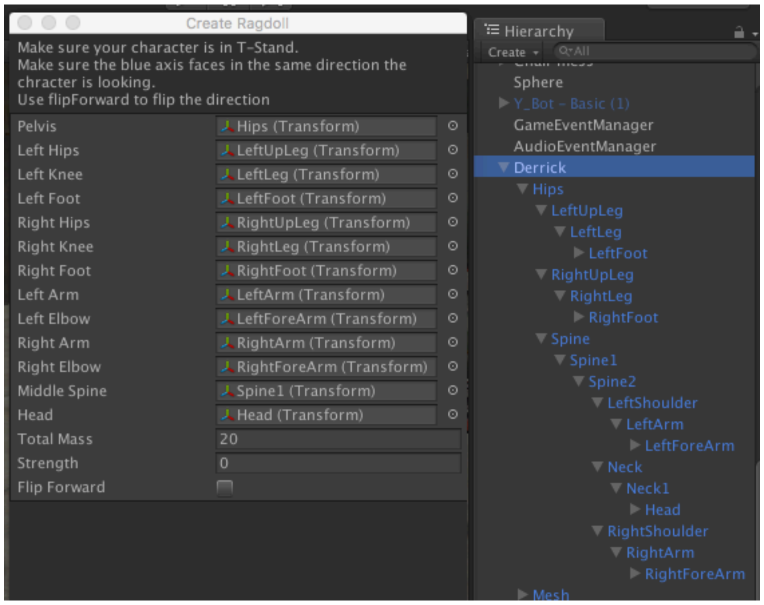

Location of “Better Ramp”

Custom Friction (Low)
Next to the box implemented from the previous task, create a green rigidbody box. Modify it to have a new PhysicsMaterial that has 0.1 dynamic and static friction and ‘Friction Combine’ set to ‘minimum’. Make sure this box slides down “Better Ramp” when the game is played.
(5 points)

Bouncy Rigidbody
Create an orange rigidbody sphere. Assign a new PhysicsMaterial that has a 0.9 bounciness and ‘Bounce Combine’ set to ‘maximum’. Place it a bit above the ground. You should see the ball bouncing when the game plays.
(5 points)

Ragdoll
Place the existing ‘Y_Bot’ asset (e.g., drag from Project Window) in the scene right above the nearest hurdle (see GameObject with name “propHurdle” as in track and field obstacle for jumping over).
(ProjectWindow:Assets/ModelsAndAnimations/YBot with Animations/Y_Bot/Y_Bot)

Run the Ragdoll Wizard via Edit Menu:GameObject->3D Object->Ragdoll…
Assign the bones as detailed in the image below. Make sure Y_Bot is aligned such that he collides/flops over the hurdle as he falls when the game starts. It is ok if the ragdoll falls off to the ground. It’s better to drag and drop directly from Y-Bot’s skeleton hierarchy rather than using the radar button to search by name when setting the Ragdoll Wizard parameters. This is because there are multiple models all using the same names and you can’t tell the difference.
(15 points)



Create Ragdoll

Make sure your character is in T-Stand.
Make sure the blue axis faces in the same direction the chracter is looking.
Use flipForward to flip the direction

Pelvis	Hips (Transform)
Left Hips	LeftUpLeg (Transform)
Left Knee	LeftLeg (Transform)
Left Foot	LeftFoot (Transform)
Right Hips	RightUpLeg (Transform)
Right Knee	RightLeg (Transform)
Right Foot	RightFoot (Transform)
Left Arm	LeftArm (Transform)
Left Elbow	LeftForeArm (Transform)
Right Arm	RightArm (Transform)
Right Elbow	RightForeArm (Transform)
Middle Spine	Spine1 (Transform)
Head	Head (Transform)
Total Mass	20
Strength	0
Flip Forward	☐

Hierarchy

```
Sphere
▶ Y_Bot - Basic (1)
GameEventManager
AudioEventManager
▼ Derrick
  ▼ Hips
    ▼ LeftUpLeg
      ▼ LeftLeg
        ▶ LeftFoot
    ▼ RightUpLeg
      ▼ RightLeg
        ▶ RightFoot
    ▼ Spine
      ▼ Spine1
        ▼ Spine2
          ▼ LeftShoulder
            ▼ LeftArm
              ▶ LeftForeArm
          ▼ Neck
            ▼ Neck1
              ▶ Head
          ▼ RightShoulder
            ▼ RightArm
              ▶ RightForeArm
  ▶ Mesh
```

<u>Scripting Forces</u>

Make a **black** click beetle or jumping bean out of a capsule. This beetle/jumping bean is ***not*** controlled by the player. Instead, it should be completely autonomous. You will need to write a MonoBehavior script that **intermittently** (random amounts of time) applies a **variable force** (random magnitude and direction) to the rigidbody in FixedUpdate(), while the jumping bean is grounded.

How this is accomplished is up to your implementation, but it is recommended to use an impulse or instant velocity change for the force type, which will give a good jump effect. Also, a torque can be applied as well. Liberal use of the Random functions is needed to make the behavior more realistic (e.g., there should be bounded but random time between jumps, jump angle and force magnitude, etc.). For the grounded requirement, look into how the ground is tagged “ground”. Furthermore, the characters have demonstrations of grounded checks.
(20 points)

<u>Make a Pause Script</u>

To better view what you implement (especially in builds), make a pause script that starts with the game in paused mode and toggles via a letter “p” keypress. Create a new Monobehaviour script with your logic and attach to an empty game object in your scene. You can pause by manipulating Time.timeScale. You can use Input.GetKeyUp() as well. This will allow your grader to unpause and observe your work after the Unity splash screen finishes. Make sure you test it out in your build. Note that this script is only needed for this milestone. You should remove it if you build upon this milestone for future assignments.

Congratulations, you have now experimented with many of the rigidbody physics simulation capabilities of Unity and are ready to apply these features to your game project!

Task Checklist

Modify the scene to include all of the following within view of the camera at start of the game. Note that this is just an overview of the tasks mentioned previously. You need to refer to the more detailed descriptions to confirm you implement them correctly:

1.) Your name appears in the HUD
2.) Vertical stack of three (3) **blue** rigidbody spheres with collision sounds
3.) Vertical stack of three (3) **red** rigidbody spheres that don't collide with one another.
4.) Asset Store model "Free Japanese Mask" with custom compound collider that tips over on nose similar to provided video link above. (DO NOT USE A MESH COLLIDER FOR THIS TASK.)
5.) yellow jointed chain made of at least five (5) rigidbody GameObjects
6.) **blue** kinematic rigidbody elevator using Mecanim animation with **red** rigidbody sphere going for a ride
7.) **green** Weeble Wobble/Punching Bozo that tilts but can't be knocked over
8.) a **purple** cube with rigidbody box that does **not** slide down the ramp
9.) **green** cube with rigidbody box that does slide down the ramp
10.) **orange** rigidbody sphere that bounces
11.) Y_Bot ragdoll that collapses over the hurdle GameObject (it's ok for ragdoll to fall off)
12.) **black** click beetle or jumping bean that jumps intermittently and is autonomous (not controlled by the player). There should be a variable/random amount of time between jumps and variable/random force magnitude and direction. Jumps only happen when grounded.
13.) Make a pause script that starts the game paused and unpauses with "p" keypress
14.) **Make sure all deliverables are observable in your build (test it!). The TAs predominantly use the build for runtime assessment. Everything should be in view of the camera.**
15.) submit project according to submission requirements (Make sure you pass audit)

Submission:

You will submit a Zip/7Zip of your project via Canvas. If the file is too big for Canvas, then submit a link to a private cloud hosting (such as GT's Box license). **Please clean the project directory to remove unused assets, intermediate build files, etc., to minimize the file size and make it easier for the TA to understand. Refer to Assignment Packaging and Submission on the Canvas Syllabus for further details.**

The submissions should follow these guidelines:

a) Your name should appear on the HUD of your game when it is running.
b) Follow the *Assignment Packaging and Submission* steps including:
    i. ZIP file: *<lastName_firstInitial>_m<milestone number>.zip*
    ii. Complete Unity Project
    iii. Builds (Mac build for both Intel 64-bit and Apple Silicon)
    iv. Readme file should be in the top-level directory: *<lastName_firstInitial >_m<milestone number>_readme.txt* and should follow base requirements from *Assignment Packaging and Submission*
    v. Size reduction

Submission total: (**up to 20 points deducted** by grader if submission doesn't meet submission format requirements)

Be sure to save a copy of the Unity project in the state that you submitted, in case we have any problems with grading (such as forgetting to submit a file we need). Do not alter or remove your submission from cloud hosting until your grade has been returned.

Resources:

Scaffolding project for this assignment:
https://github.gatech.edu/IMTC/CS4455_M1_Support

Collision Action Matrix – This is very useful for understanding Unity's collision rules
https://docs.unity3d.com/Manual/CollidersOverview.html

Event System Tutorial - https://unity3d.com/learn/tutorials/topics/scripting/events-creating-simple-messaging-system

Also, be aware that the tutorial's EventManager has been improved and can be found in the M1 project resource.