Modern Computer Games

COMP 521, Fall 2020

Assignment 2

Due date: Thursday, October 22, 2020, by 6:00pm

Note: Late assignments will only be accepted with prior **written** permission of the instructor. You must **explain** all answers and **show all work** to get full marks! Please make sure your code is in a professional style: **well-commented,** properly structured, and appropriate symbol names. Marks will be very generously deducted if not!

Description

Note that this assignment requires you build a scenario with 2D game mechanics. **This must be done in Unity, using the 2D asset package provided in MyCourses.**
Note that in this assignment **you must do your own collision detection/resolution and manage your own physics**. You may continue to use basic primitives for distance computations, line/ray-casting, and intersection between points, lines, and simple objects (e.g., rectangles, triangles, circles).

1. First, you need to produce a game terrain. This will be a 2D profile of a two small mountains with a water-filled valley between them, leaving a relatively flat space on either side, as shown below. **10**

   The ground/mountain-line and water-line shape should be randomized for each playthrough, but still overall similar: use 1D Perlin noise to alter a fixed starting outline, giving it both coarse and fine-grain detail while retaining the overall general shape. Note that you must compute your own Perlin noise—do not use built-in functions or assets.

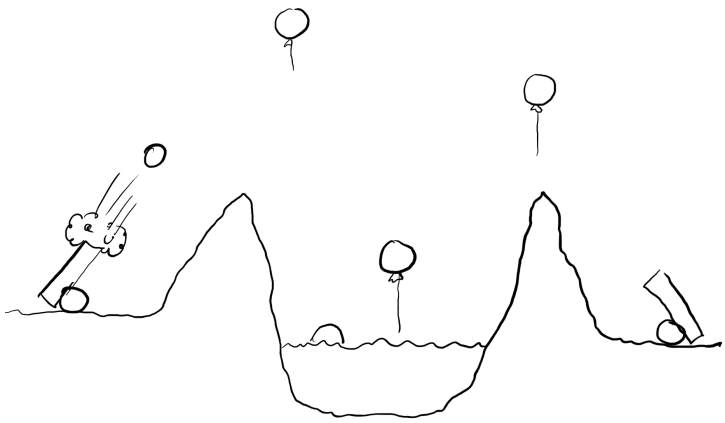

Figure 1: Sketch of terrain structure. Cannons not to scale.

2. On the right and left sides of the screen are player-controlled *cannons*. Each is generally pointed toward/over the mountain. Your cannons do not have to be nicely drawn, but should have a recognizable barrel (rectangle) that correctly indicates the current angle of fire. **10**

   The cannonballs emitted as a result should be drawn as small circles, with motion modelled using projectile physics, incorporating initial velocity, barrel angle, gravity. Wind will also be required (see below), although it does not affect cannonballs. You do not need to model wind resistance.

   Determine appropriate gravity and a range of barrel velocities (assume cannonballs have unit mass). It should be possible with some combination of angle and velocity for cannonballs to hit the mountain in front of them, land in the water, or hit the mountain or flat area on the other side. Ensure the cannonball motion is easily visible to a human observer.

Pressing the tab-key should toggle which cannon is being controlled, and pressing the spacebar fires the cannon. Barrel elevation should be controlled within a 90° range (ie ranging between horizontally pointed toward the other cannon and pointing straight up), increasing with the up-arrow and decreasing by a down-arrow press, while muzzle velocity is increased/decreased by left/right arrows. Ensure the current muzzle velocity is also represented in some fashion.

炮弹初速度

3. Cannonballs that go offscreen left or right, land in the water, or end up stationary for an extended period should disappear. Cannonballs also do not interact with other cannonballs or the cannons themselves. Cannonballs that encounter the mountains or ground, however, should collide with the surface. For this you must implement your own collision detection and handling (do not use any built-in physics or colliders—you may, however, use basic geometric primitives for primitive shape intersection, distance, etc, as well as raycasting). **15**

   The exact parameters of your collision resolution are to you, but there should be some reasonable "bounce" to a cannonball/surface collision (ie, a coefficient of restitution of between 50% and 95%).

   Note that you do not need to model any rotational effects on the cannonball collisions.

4. Rising out of the water are some number of balloons, represented by lines and points. Balloons are spawned at random locations within the watery area at about one per second; they should slowly rise straight up, taking 3–5 seconds to reach the top. A balloon that goes off-screen (whether left, right, or top) de-spawns. **15**

   Balloons and their trailing strings (see image below) must be modelled as points and constraints between points, with all motion based on a Verlet integration strategy. You need to decide on appropriate constraints between points: your balloon can have some flexibility in shape and need not stay circular, but should always be a simple polygon; the string has more freedom, and may cross itself, but should not end up inside the balloon. Balloons and their strings do not interact with other balloons or their strings in any way, but neither they nor their strings should intersect the mountains.

   Implement all behaviour of the balloons as Verlets, using your own implementation of Verlet integration and collision and constraint solving. Provide as a *separate document* (.pdf) an annotated drawing indicating what constraints you added to the Verlet-balloon to ensure its shape.

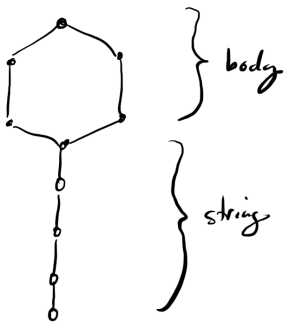

Figure 2: A balloon. You can vary the design and add additional points for helping enforce constraints, but it must minimally include at least 5 points forming the ballon itself, and at least 4 points forming the string (not including the point connecting it to the balloon).

5. Cannonballs that intersect a balloon destroy the balloon, but are not otherwise affected by the encounter. A cannonball does not destroy a balloon if it only encounters the balloon string, but the string should not intersect the cannonball. **5**

6. Above the mountain tops (but over the whole scene width) wind exists, and affects balloons as they rise out of the valley. Determine the maximum height of your mountains, and apply wind force to any balloon point that goes above. Model wind as a randomly changing horizontal force that pushes on (some part(s) of) the balloon itself (but not the string). **5**

balloon will be deformed a little bit, since each point will be affected by the wind at different time

Determine a reasonable range of magnitudes $[-w \ldots w]$ representing left→right movement if $w > 0$ and right→left if $w < 0$, such that a wind of magnitude $|w|$ can push a balloon off the left/right sides of the screen before it floats to the top.

Wind force should change randomly within your range every 2s. Include some representation of relative wind speed and direction, either graphically or through an onscreen text display (see asset package).

What to hand in

Assignments must be submitted on the due date **before 6pm**. Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**.

Include all source code necessary to run your simulation.

TODO:
-balloon collision with mountain
-string intersection with balloon body

This assignment is worth 15% of your final grade. **60**