Modern Computer Games
COMP 521, Fall 2021

Assignment 2

Due date: Friday, October 22, 2021, 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. To make it easier to draw lines, points and text in Unity, an optional 2D line-drawing asset package is provided in MyCourses. Note that in this assignment you must do your own collision detection/resolution and manage your own physics. All object movement and collisions should be done entirely by code you write! You may, however, continue to use any existing, built-in primitives for doing actual intersection calculations between primitive geometric objects/shapes (points, lines, rectangles, triangles, circles). Do not use the Unity physics (colliders) to detect or respond to collisions.

1. First, you need to produce a game terrain. This will be a 2D profile of an uneven, but more-or-less flat terrain, with a central mound, as shown below, and bounded on the left by a vertical wall.

   The ground (which includes the central mound) texture/shape should be randomized for each playthrough, but still overall similar: implement and use either midpoint-bisection or 1D Perlin noise (do not use the built-in Perlin noise function) to produce a profile that has both coarse and fine-grain detail while retaining the overall general shape.

   ![Terrain Sketch]

   Figure 1: Sketch of terrain and level design.

2. On the right side of the screen is a player-controlled cannon, directed toward the left side of the screen. Your cannon does not have to be nicely drawn, but should have a recognizable barrel (rectangle) that correctly indicates the current angle of fire.

   The cannon can be fired, emitting cannonballs that should be drawn as small circles, with motion modelled using projectile physics, incorporating initial velocity, barrel angle, gravity. Cannonballs are not affected by wind, and you do not need to model wind resistance, but do include some amount of friction so they eventually stop moving.
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 mound directly, and to shoot up and over it to land on the terrain on the other side of the mound. Ensure the cannonball motion is easily visible to a human observer.

Pressing the spacebar fires the cannon. Barrel elevation should be controlled within a 90° range (ie ranging between pointing horizontally 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.

3. To the left of the cannon is an invisible wall. Cannonballs that go offscreen in any way, or which contact the invisible wall from the left side, or which end up stationary for an extended period should disappear. Cannonballs that encounter the mound, ground, or other cannonballs, however, should collide with some reasonable response. For this you must implement your own collision detection and handling (do not use any built-in physics or colliders).

The exact parameters of your collision resolution are to you, but there should be some reasonable “bounce” to a cannonball 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, nor transfer of momentum due to cannonball collisions.

4. On the left side of the terrain are some number of insects as shown below. Insects are spawned at random locations left of the mound, and gently float around using just random motion. Insects do not interact/collide with each other, and may thus overlap. Any insects which encounter the invisible wall in front of the cannon or which go offscreen at the top should be despawned, and replaced by a new, fresh insect.

Insects should be entirely modelled using Verlets. You will need to decide on appropriate constraints between points: insect wings and antennae should have significant flexibility, and may briefly fold or self-intersect, but should generally be able to return to its original shape.

Be sure to use your own implementation of Verlet integration including collision detection and constraint solving. Provide as a separate document (.pdf) an annotated drawing indicating what constraints you added to the Verlet-insect to constrain its shape.

Figure 2: An insect. You can vary the design and add additional points for helping enforce constraints, but it must minimally include at least 4 points forming each wing, 1 point for the body, and 2 points to form the antennae.

5. Insects can collide with the ground and cannonballs—no collision response is required but they should not interpenetrate. Cannonballs are not themselves affected by collisions with an insect.

6. Left of the mound add 3 sources of rising air. Each source generates a vertical column of air moving upward at a random speed. The air-speed of a column changes (instantaneously) about every 2s, and should range between 0 and whatever speed would push a point from the bottom to the top of the scene in about 0.5s. This rising air does not affect cannonballs, but does affect insects (or rather, whichever vertices of an insect that are inside the column). As with other insect motion the effect on insects should be modelled using Verlets.

Include some representation of relative air-speed for each column, 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.

This assignment is worth 15% of your final grade.