Modern Computer Games  
COMP 521, Fall 2019  

Assignment 2

Due date: Wednesday, October 23, 2019, 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. You may continue to use Unity, but will need to enforce the 2D-ness yourself. You may also do this assignment in Java, Python or HTML5/JavaScript if you prefer, but you must then include a readme.txt file describing how your code is structured and how to run it.

Note that whatever your implementation context, you must do your own collision detection/resolution and manage your own physics. You may, however, use basic primitives provided by your environment for distance computations, 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 flat terrain, with a stonehenge in the centre, as shown below.

   The stonehenge overall shape is constant, but the texture of it should be slightly randomized to be different on every play-through. Use 1D Perlin noise to alter a starting outline, giving it both coarse and fine-grain detail while retaining the overall general shape.

   Figure 1: Idea for basic shape.

2. On the right side of the screen is a player-controlled cannon. It is generally pointed toward/over the stonehenge. Your cannon does not have to be nicely drawn, but should have a recognizable barrel (rectangle) to indicate the current angle of fire.

   The cannonballs emitted as a result should be drawn as small circles, with motion modelled using projectile physics, incorporating initial velocity, barrel angle, gravity, and wind (you do not need to model wind resistance).

   Determine appropriate gravity, barrel velocity (assume cannonballs have unit mass). Absent wind, the goal is that your cannonballs should be able to get over the stonehenge and reach the other side at a reasonable (45°–60°) launch angle. Your cannonball flight time should be very apparent to the user, taking at least 0.5s of real-time for a cannonball to reach the other side.
Wind force is only applied above the stonehenge, in a horizontal direction. Determine a reasonable range of magnitudes $[-w \ldots w]$ representing left→right movement if $w > 0$ and right→left if $w < 0$, such that when at $w$ the wind force slows the cannonball sufficiently to prevent it from reaching the other side of the stonehenge but does not blow it backward, given the above default configuration of other parameters.

Wind force should change randomly within your range every 2s. At the top of the screen should be one or more cloud(-like) objects that move in sync with the wind, to clearly indicate the wind direction and magnitude. Note that a cloud should always be visible (either by wrapping or using multiple clouds).

Pressing the spacebar fires the cannon. Barrel elevation should be controlled within a $0^\circ$–$90^\circ$ range (pointed left and up), increasing with the up-arrow and decreasing by a down-arrow press.

3. Cannonballs that go offscreen or land on the flat ground disappear. Cannonballs that encounter the stonehenge, 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).

For simplicity, cannonballs do not collide with other cannonballs, the cannon, or the cloud(s). A cannonball that loses all motion for any reason should be destroyed after a few seconds.

The exact parameters of your collision resolution are to you, but there should be some reasonable “bounce” to a cannonball/wall 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. The left side of the side shows 4 small “ghosts”, represented by lines and points (and larger points or circles for eyes). Ghosts are spawned at random locations left of the stonehenge; they should slowly rise up above the stonehenge and drift to the right in order to get to the right side of the screen (where they de-spawn).

Ghosts 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, aiming to give the ghosts a flexible, somewhat floppy appearance, while still avoiding gross distortion (eyes should stay inside, and while the bottom trails may sometimes briefly twist or overlap the overall shape should usually be recognizable).

Provide as a separate document an annotated drawing indicating what constraints you added to the Verlet-ghost to ensure its shape.

![Figure 2: A ghost. You can vary the design and add additional points for helping enforce constraints, but it must minimally include a head with two eyes and at least 3 non-zero width trails coming out of the body.](image)

Ghosts do not interact with other ghosts, wind, the cannon, or the cloud(s), and like cannonballs disappear if they exit the scene. Ghosts do not have collision response with the stonehenge or ground, but should not interpenetrate (or end up inside the stonehenge).
A ghost that disappears for any reason should result in another ghost spawning at a random location left of the stonehenge.

5. Cannonballs are not affected by ghosts, but ghosts are affected by cannonballs. Upon contact with a moving cannonball the velocity of an incoming ghost is added to whichever vertex or vertices intersect the cannonball, and the cannonball disappears.

**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.

For non-coding questions, submit either an ASCII text document or a .pdf file with all fonts embedded. Do not submit .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.

This assignment is worth 15% of your final grade.