Modern Computer Games

COMP 521, Fall 2018

Assignment 2

Due date: Wednesday, October 24, 2018, by 6:00pm

Note: Late assignments will only be accepted with prior **written** permission of the instructor. 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

This assignment requires you build a scenario with 2D game mechanics. You can 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 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 small mountain, reaching about 1/2 up the screen, and separating the left and right halves of the screen. On the extreme left is a full-height wall, to prevent anything from exiting in that direction. **10**

   The mountain shape should be randomized, different on every play-through. The profile should include both fine-grain and coarse levels of detail, using either midpoint-bisection or 1D Perlin noise to produce the visual representation. Below is shown a schematic view (without any detail on the mountain).

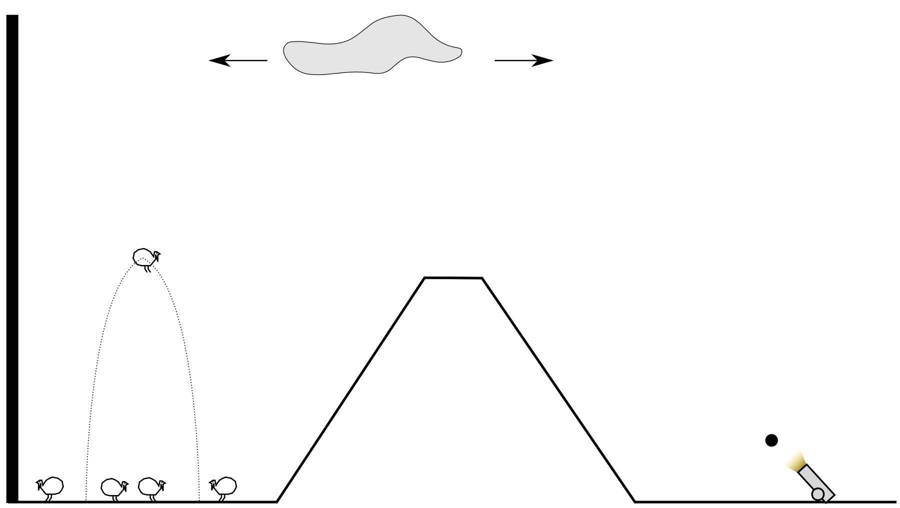

Figure 1: Overall view of the environment.

2. On the right side of the screen is a player-controlled *cannon*. It is generally pointed toward/over the mountain. Your cannon does not have to be nicely drawn, but should have a recognizable barrel (rectangle) to indicate the current angle of fire. **10**

   The cannonballs emitted as a result should be drawn as recognizable 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 mountain and reach the other side of the mountain at a reasonable

(45°–60°) launch angle. Your cannonball flight time should be 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 mountain top, 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 prevents the cannonball from reaching the other side of the mountain but does not blow it backward, given the above default configuration of other parameters.

Wind force should change randomly within your range every 0.5s. 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.

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

3. Cannonballs do not interact with the cannon, other cannonballs, or the cloud above. A cannonball that encounters the ground, stops moving, or manages to exceed the left/right bounds of the screen disappears. **15**

   Cannonballs that encounter the mountain or the wall, however, require collision handling. For this you must implement your own collision detection and handling (do not use any built-in physics or generic colliders—you must use only basic geometric primitives for primitive shape intersection, distance, etc. Line/raycasting is also ok).

   The exact parameters of your collision resolution are up to you, but there should be some reasonable "bounce" to a cannonball – mountain/wall collision (e.g., a coefficient of restitution in [0.5,0.95]). You do not need to model any rotational effects or friction.

4. On the left are small, stick-figure animals, let's call them turkeys. These should be modelled using Verlets. You will need a point/line model of an animal such as shown below, moving each point separately according to Verlet integration, while ensuring there is some effort to retain the overall shape and avoid gross distortions (but also not being overly rigid). **15**

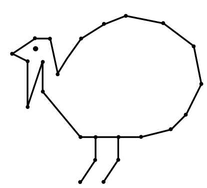

Figure 2: A "turkey." You do not need this exact number or arrangement of points/lines, but it should have a body, head, beak, eye, 2 legs, and a wattle.

   Turkeys slowly pace (slide) back and forth between the wall and the mountain start on the left. Every few seconds a turkey may suddenly leap straight up, peaking just slightly higher than the mountain top before falling back down.

   They do not collide with each other, but may be affected by wind to the extent they go above the mountain top. They do not have collision responses with the surrounding terrain, but should not penetrate the mountain, wall, or ground. A turkey that lands on the left side of the mountain should start moving to the left to eventually reach the ground. A turkey that somehow ends up on the right side or otherwise out of bounds should disappear and be re-initialized on the left.

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

   Populate the left side with 5 turkeys, starting at random positions and moving in random directions, but at the same speed.

5. Cannonballs are not affected by turkeys, but turkeys are affected by cannonballs. Upon contact with a moving cannonball the velocity of an incoming turkey is added to at least one of the turkey vertices, and the cannonball disappears. **10**

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. $\overline{\textbf{60}}$