Realtime Rendering

Assignment 3

Assessment: 33%

Deadline: Fri 13/10/2017 9pm (end week 12)

Changes and Clarifications:

- None so far

Aims

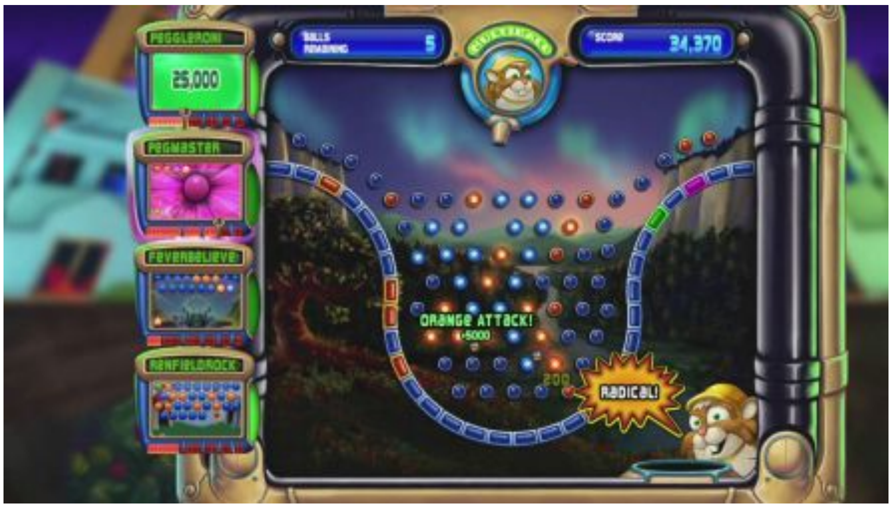
This assignment is intended to have students learn about game physics and collisions, and to write a small game using mechanics based on such. It may be done in pairs.

Problem Description

Many games have been written which have 'game mechanics' based on relatively simple physics and collisions, and were core to many early games - including one of the very first computer games Spacewar!. Highly successful games based on simple physics continue to emerge. Osmos, which uses inelastic collisions between circular/spherical objects. Angry Birds (and locally produced Catapult King by Wicked Witch Software might be of interest), uses projectile motion and collisions.

This year the final assignment is based on Peggle which uses projectile motion and collisions.

The assignment is to write a game similar to Peggle, involving collision detection between a moving ball and fixed position pegs. However, there are variations including making the pegs shapes be different polygons - triangles, squares, pentagons etc - and be rotating.



(from wikipedia) And [youtube video of gameplay]

Details

Start with elastic 2D collisions amongst circles/disks in a rectangular arena.

A basic assignment (CR level) should include the following:

- Implement a rectangular arena where the ball collides elastically with walls and pegs.
- The pegs should light up when hit.
- The ball should fall with a constant acceleration towards the bottom, and either fall out of the bottom of the arena or be caught by the catcher.

- The trajectory the ball is predicted to follow should be visualised, including collisions. For gameplay this should be limited two deep.

  There should also be a mode which shows the complete *predicted* trajectory of the ball. However, the actual trajectory will deviate from it due to limitations of numerical integration, even if higher precision (double, long double), and/or better numerical methods are used. (And is why for example weather forecasts are wrong, and limited to 7 days). For the assignment purposes just show the initial predicted trajectory.

- Use simple plain colours for rendering, including the visualisation.
- Use brute force collision detection whereby all pegs are checked.

There are some features (broadly speaking, for DI level) which should be considered first once a basic assigment as above has been completed,

- Polygonal pegs: rectangles, triangles, pentagons and hexagons.
- In the visualisation include highlighting of the edge of the polygonal peg the ball bounces off
- Handle collisions with corners (vertices) by assuming circular shapes.

There are many further extensions possible (for HD level), and this assignment is more open ended with respect to them than the previous assignments.

- Use a *core* OpenGL context which means compatibility and "boots and all" including deprecated features are not allowed. No immediate mode.
- Use a core OpenGL ES 3.0 context, which is used in many mobile devices, and the basis for WebGL 2.0, but can be used on the desktop too. Use SDL and target an Android phone with ES3 support.
- Particle effects.
- Add moving features, which may collide.
- Improve the collision detection performance by using a uniform grid.
- Implement different shaped arenas, e.g. a circular arena/world.
- Allow different arenas or features made of bricks which get knocked out.
- Using shaders to render the ball, pegs and bricks, treating them as either quads or points (with point size). Further, use a fragment shader to apply an animated procedural texture.
- Explore deeper game design changes, e.g. adding sections where pegs are not fixed but moving and colliding, or where pegs are rotating randomly
- Use analytic cirle-parabola collision detection for trajectory prediction.
- Other features or improvements of your own. Best to discuss with Geoff first.