

University of Colorado  
Department of Computer Science  
Computer Graphics – CSCI 4229  
Spring 2003

Problem Set 13

Issued: 29 April 2003

Due: code part 5 May 2003, 5pm; written part 2 May 2003, 5pm.

1. Do exercises 7.4, 7.8, and 7.15 on pp364-365 of the textbook. Please submit your answers in writing, either in the box outside my office door (ECOT 524), or in the CS department office (7th floor).

**The rest of this problem set is optional.**

If you choose to do it, I will drop the lowest *two* problem set scores (three if you take quiz III). If your game is truly impressive, I will be correspondingly impressed—and free with extra credit.

There are two options, both involving simple videogames that build upon material covered in the previous problem sets. You may pair up if you wish, but please contact me ahead of time if you want to do so. Paired project teams will need to implement a more-demanding version of the game, as described below.

Please mail the tarball of code solutions, in the usual format, **to me, not the grader**: `lizb@cs`. The deadline is firm, as grades have to be in by noon on 7 May.

• • •

*Choice I:* A traditional single-player videogame, where a creature with bad intentions chases the user's avatar around a 3D virtual environment. This is based on the virtual world that we have been building up through the problem sets. You may comment out the embellishments added during the last few weeks—like the images, textures, funky lighting, and furnishings—if they make things run too slowly. I will uncomment them and run your full-bore game on a faster machine for evaluation, so please put appropriate explanation in comments—e.g., “uncomment this block to enable wall texturing...”

1. Create some interesting creature and have it fly around randomly<sup>1</sup> in the PS12 world. This creature need not obey gravity, and it may fly through walls for this part of the problem. Allow the user to choose the creature's maximum flight speed at the beginning of the game, via a command-line interaction with the user; this amounts to a primitive kind of “difficulty level” setting. Add some sort of bounding box check in order to keep your creature from flying off to infinity.
2. Modify your creature's path-planning algorithm to have it chase you (i.e., your avatar). Assume that it knows your position even if there is no line-of-sight path between you and it. This is easy if the creature can fly through walls and hard if it can't (see below).
3. Give your creature a weapon that it shoots at you at random intervals *when it has a line-of-sight path to you*—that is, when there are no walls between it and you. You need not show the actual weapon nor model gravity's effects on the projectile, but you should show the projectile's flight somehow. Modify your avatar code to do something appropriately theatrical when it is struck by a projectile.

*For paired projects:*

- Defend yourself. This can include passive (i.e., shield) and/or active (i.e., shooting back) devices. Choose your design parameters so the contest is even—e.g., limit your firing rate, have the shield pin you in place when invoked, etc. Please explain your scheme *and its interface* in the comment block at the top of your code.

---

<sup>1</sup>random small direction and speed changes to its current trajectory at every step

- Add gravity and a floor to your game, and make all movement (you, creature, projectile) physically realistic. Among other things, this means that projectiles follow parabolic paths, critters and avatars are pulled back to the floor by a constant force, etc.

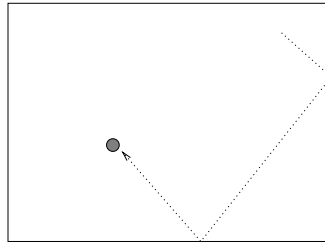
If you want to do anything interesting with these facilities—e.g., allow avatars and creatures to jump, invert gravity briefly, etc.—that’s fine, but please explain in the comment block.

- Modify the bad guy’s path-planning algorithm so that it is *unable* to walk through walls, but still able to chase you effectively.

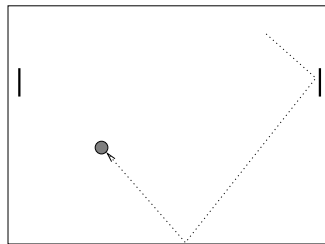
• • •

*Choice II:* A 2D Pong game where two players hit a bouncing ball with a paddle.

1. Draw a rectangular box on the screen and bounce a ball around in it at constant speed:



2. Modify your code to add paddles (short vertical line segments) on either side of your box, as shown below. Allow the user to move each of these segments up and down using some pair of keys. I suggest that you choose key pairs that are not adjacent, lest your players end up elbowing one another.



Make sure that your code handles bounces off the paddles properly. Assume that the collisions are inelastic and that the paddle does not recoil. This means that the ball bounces instantaneously, flying away at the same velocity at which it came in. Don’t worry about tangential forces of moving paddles — that is, adding “english” to the ball.

3. Add goals to your problem: gaps in the end walls that the adjacent player defends (and the opposite player aims for). Your code should detect when the ball passes through this gap, perform some appropriate pyrotechnics to reward the person who shot it through, and restart with a new ball at some random position and direction.

4. Make the ball’s dynamics physically realistic by incorporating gravity into its vertical velocity. (The horizontal speed should remain constant.)

5. Now let the user move the paddles *horizontally* as well, and modify the ball trajectory mathematics to act in an appropriate way (i.e., include the extra normal force imparted by a horizontally moving paddle, which will change the horizontal component of the ball’s velocity).

*For paired projects:* make the paddle/ball *collisions* physically realistic with respect to tangential forces. This is open-ended — and much harder than the rest of the problem; if you solve it, your users will be able to put “english” on the ball. You will have to make some assumptions about friction between the ball and the paddle in order to do this, and you should render the ball such that the spin is visible as it moves through space.