Assignment 2 – 2D game with collision

Familiarize yourself with vertex arrays, shapes and the lab exercise on time management. In this project it's important to manage the time and motion of all moving objects in the way described there.
The assignment is to implement a 2D computer game that uses vectors to move objects and collide/bounce them off of each other.
First there are a few definitions/explanations worth reviewing.

Collision detection

Check if an object has crossed a boundary.

Simple – static check for single point crossing vertical line
Check on which side the point is at beginning of frame (compare x-values)
Add to the point how much it will move horizontally this frame
Check if point is on other side of horizontal line after that amount of movement
If so, collision happened!

More complicated – dynamic check for single point crossing diagonal line
Use formula for intersection of ray with line (explained in "Examples 1" on lecture page)
When t.hit is found, check if that time is within the current frame (between 0 and deltaTime)
If so, check to see if P.hit is between the end points of the line
If so, collision happened!

Collision responce

Change position/direction/speed of object – bounce!

Simple – bounce off a horizontal or vertical obstacle
Flip the relevant coordinate of your motion/speed vector.

More complicated – bounce off a diagonal obstacle
When looking at projections of vectors onto lines we derived a formula for reflection
First normalize (to length 1) the normal (n) to the line you're bouncing off of, then
r = a – 2(a (dot) n)n, where
a is the vector the object is travelling along initially,
r is the vector the object is travelling along after the bounce and
(a dot n) is the dot product of ***a*** and ***n***

About drawing several different objects

We have looked at seting up a single vertex array for each object type/shape in a lab exercise and to separate the code for each into different classes for simplicity. There are more ways to do this. All objects can be put into a single vertex array or a vertex array made for each object instance. Make a decision on what works best for each object type in your game.
Just remember to call ***glVertexPointer*** before calling ***glDrawArrays*** to point OpenGL to the right set of points.
For circles and rectangles it‘s probably best to have a single vertex array for the shape and then use translations and scaling to draw each instance.
However, for the diagonal lines in the cannon program (for example) it's probably best to make a new vertex array for each line and simply not use glTranslate, glRotate and glScale. That way it‘s simpler to keep track of the end positions.

Assignment requirements in general

The assignment must include:

- An object controlled by the player
  - Cannon, paddle, spaceship
- An object that moves on its own
  - Cannonball, asteroid, ping-pong ball
  - Can move continuously or be set into motion by player
- Objects that collide with other objects
  - Boxes, lines, asteroids
  - Can be added by the player or be part of a „level“
  - Collisions must be calculated
  - Collision response, either bounce or trigger event (victory, death)

Game ideas

Following are ideas for some specific games. The first idea is more detailed than the others and you can use it to measure the complexity required. You can mix any of these ideas or come up with new ones, but make sure they include the type of motion and calculations required. If in doubt ask the teacher and get help preparing a reasonably sized assignment.

Bouncing cannonball game

At the bottom of the screen draw the “cannon”, pointing up.
Somewhere in the game area draw a goal.
Using the right and left arrow keys on the keyboard the user can rotate the cannon in order to aim.
When the user hits 'z' the cannon fires a cannonball in the direction the cannon was pointing.
Once the cannonball leaves the game area or enters the goal the cannon can be fired again.

Draw a circle for the cannonball (use TRIANGLE_STRIP)

Between firing the cannon the user can add obstacles to the game area (obstacle field). Different versions of the obstacles are described below. When the cannon is fired the cannon ball interacts with the obstacles before either leaving the playing field or entering the goal.
After the cannon ball has finished its run remove all obstacles from the obstacle field.

For the sake of interaction with obstacles you can use the center of the cannon ball as a single point. You don't have to imlement the ball's size in the interactions. You still have to draw the cannonball as a circle, not a point.

Make sure the cannon, cannonball, goal and obstacles are different colors. The obstacles can all be the same color, just different from the other objects.

Simplest obstacle field (graded to 6.0, plus extras)
Before firing the cannon the user can click any number of times on the empty area of the window to add a rectangular box at that spot. The boxes can all be the same size and when tested they will not be put too close together so more than one bounce shouldn't happen within a single frame.
When the cannonball is travelling through the field it should bounce off the sides of the boxes it hits. Make sure to implement both horizontal and vertical collision.

More complicated obstacle field (graded to 8.0, plus extras)
Before firing the cannon the user can click any number of times on teh empty area of the window. Each two clicks will be interpreted as two ends of a line, that should then be drawn between them. This way many lines can be drawn.
When the cannonball is travelling through the field it should bounce off the lines as if they were walls. It should bounce off either side of the lines and never go through them.

Most complicated obstacle field (graded to 10, plus extras)
Before firing the cannon the user can click-drag-release with either the left or right mouse button. If the left button is used the click will save a corner in a rectangular box and the release will save the opposite corner. The box should be drawn while it's being made and after it's saved as well.
If the right button is used the click will save an end point of a line and the release the other end point. The line should be drawn while it's being made and after it's saved as well.
This will give us an obstacle field with different size boxes (rectangular and parallel with axis) and lines (diagonal).
When the cannonball is travelling through the field it should bounce off of both the boxes and the lines, implementing both types of collision.

Extras

No matter which type of obstacle field you implement you can add extras. Implement the simple obstacles well and add some nice game logic and graphical displays to it and it could well be an 8. The more complicated ones can both give full marks and it doesn't stop there. Extras added onto a well implemented base program can take you above 10 out of 10. Yay!

Make a game out of it - fun and as easy/difficult as you choose

Add levels, where each level has initial obstacles (that don't disappear) preventing you from shooting straight into the goal. You will need your obstacles to finish a level.
Add resources and a way to show them to the user. You have 3 boxes and 1 line to finish this level. Is a level harder because of different initial obstacles or can it be the same level, just with less resources?
Score, lives, game over...?

Iterate the collisions to allow for multiple collisions within frame - a bit of math

If an obstacle is hit in a frame, when in the frame did it happen? Let's say t.hit is 0.3 and the delta time this frame is 0.5. What happens to those 0.2 time units from the collision to the end of the frame? We should actually change the direction of the cannonball and move it by 0.2 in the new direction from the collision point P.hit. That's where it should end up this frame! What if another collision happens whithin those 0.2 time units? Repeat the collision process until no collision happens, but only using the rest of the delta time and starting from the last collision point.

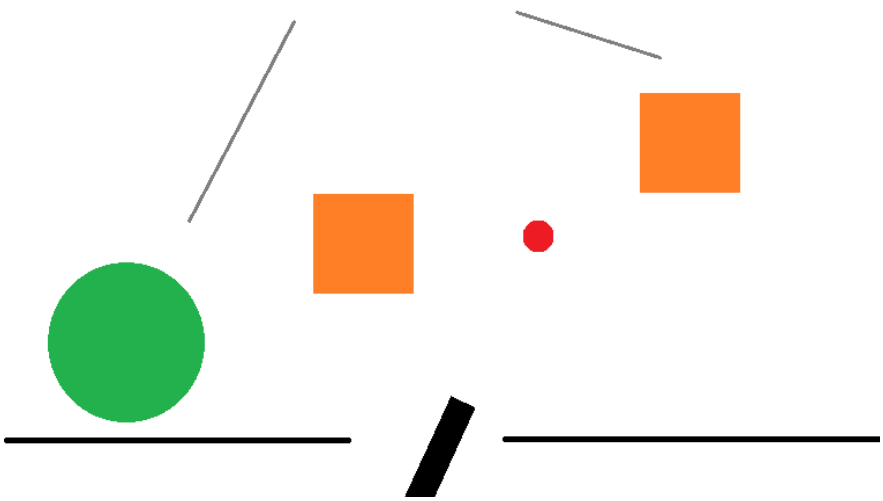
Implement the collisions using the cannonball's size – technically difficult

Here you have to add the radius of the ball to the collision detection in each direction. Try it first on the rectangles. What happens on corners? The collision responce should use a normal to the circle for bouncing is some cases. Then try the diagonal lines. You're no longer checking for a ray crossing a line, but for a ray coming within a certain distance to the line. Can you get the ends right too?

Get creative – do whatever you want

Fun surprises are always excellent.

Finally here's an example of what this might look like. In my case the cannon is black, the cannonball red and the goal is green. The black lines around the cannon are not necessary but it might be nice to have a little area at the bottom to distance the cannon from the collision area.



Breakout

The upper half of the screen contains rows and columns of rectangular bricks. At the bottom of the screen there is a paddle, controlled by the player. A ping pong ball moves continuously, bouncing off the bricks and paddle. When it hits a brick the brick should disappear. When it hits the paddle the position on the paddle (and the angle the ball is travelling at before the bounce?) should control the angle the ball travels at after the bounce. When all bricks have disappeared the game/level is won. If the ball reaches the bottom of the screen the player loses (a life?).

Here different complexities of collisions, control, levels and game mechanics give extra points, like in the Cannonball game.
For full marks make sure the program always knows which brick is hit and bounces accordingly.
You can also mix and match this with the cannonball game, using a paddle rather than a cannon to reach the goal (goals?).

Gravity sling-shot

The player uses a sling-shot (or cannon if preferred) to launch an object into space. The space includes planets with gravity (and obstacles to bounce off of?) and a goal to hit. Use the gravity of the planets to navigate to the goal without hitting the planets themselves.

Anything goes (almost)

Come up with other game ideas and find ways to add acceleration, speed boosts, gravity and bouncing to it. Pin-ball, asteroids, platform jumping.

Have fun!