CSci 1113, Spring 2018
Lab Exercise 13 (Week 14): Graphics part 2

It's time to put all of your C++ knowledge to use to implement a substantial program. In this lab exercise you will construct a graphical game that employs many of the object-oriented techniques presented in class. If you have not completed the last from last week, you should work on that and can get it checked off during this lab.

Warm-up

For the warm-ups, we will extend the game to include two objects inside the square (or "board"). To do this we will make use of inheritance.

1) Making a parent

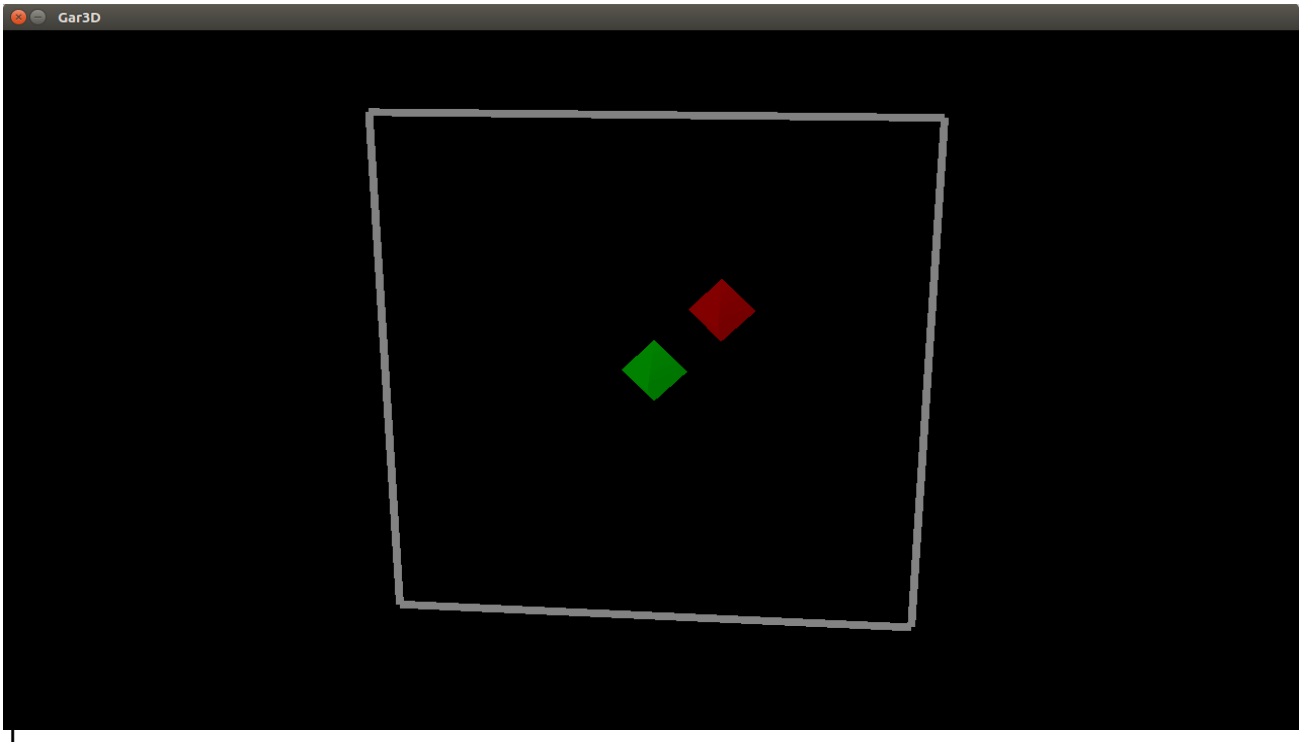
First make another class to represent all the moving objects inside the game. As this class represents all moving objects, you should put the (x,y, speed) variables inside this parent class.

You should also have a copy of the movement() and draw() functions. Both of these functions will be redefined in the child class(es), so the movement function can be blank (right now it is just a placeholder). The draw() function can draw the basic shape of the player, but either pass in a color or use a variable in the parent's draw function.

You should then modify the original class to inherit from this new parent. Change the code to reuse as much of the parent's code as possible

2) Making an enemy

Make another child of this parent class to represent the enemies in the game. These are movable objects, so they should inherit from the same parent class as the player. However, they should appear a different color to differentiate them from your player. Right now the enemy's does not need to move. Make an enemy at (0.5,0.5) with speed (0.5) and you should get:



Stretch

1) Charging enemy
Modify the enemy class to move towards the player. To do this find the x-distance and y-distance between the enemy and player. Then choose whichever distance is greater and move in the direction to shrink the distance (in other words, the enemy moves towards the player). If there is a tie, you can pick either direction. Assume the player moves first and then the enemy reacts to this movement. The enemy should only move if the player moves. (Note: if the enemy is on top of you, do not have the enemy move.)

Hint: You may want to redefine how the movement() function works for moving objects, players and enemies.

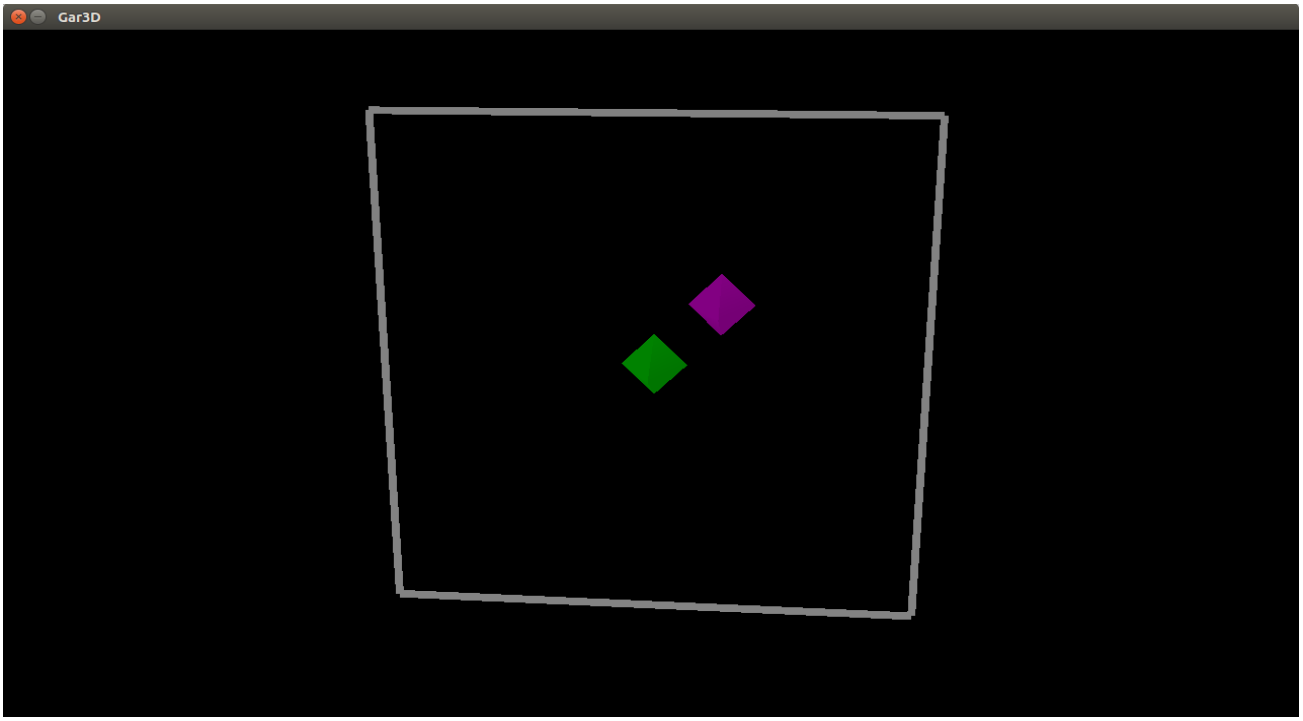
2) Losing the game
If the enemy has either an x-distance or y-distance of less than 0.5, you should stop the game, as you have lost. This means the enemy must have moved into the same space as you (the game is a 9x9 grid if you followed the directions with 0.5 speed and walls at 2 and -2).

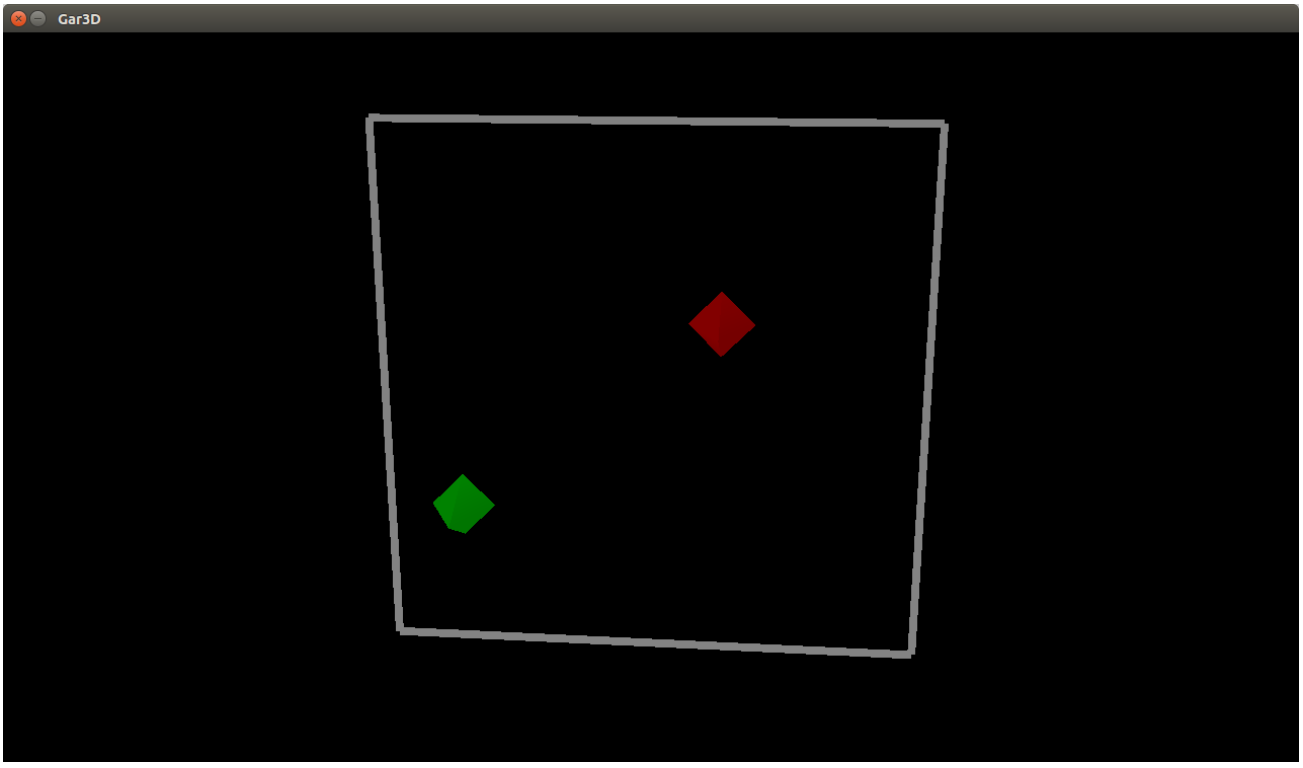
Also, remove the old winning condition for getting to the top left. (Sadly you can only lose with these changes.)

Workout

1) Growing enemies
Modify the enemy class to have them start in a weakened state. (Represent this by a different color.) If you move on top of a weakened enemy, the enemy dies. Weakened enemies should not move.

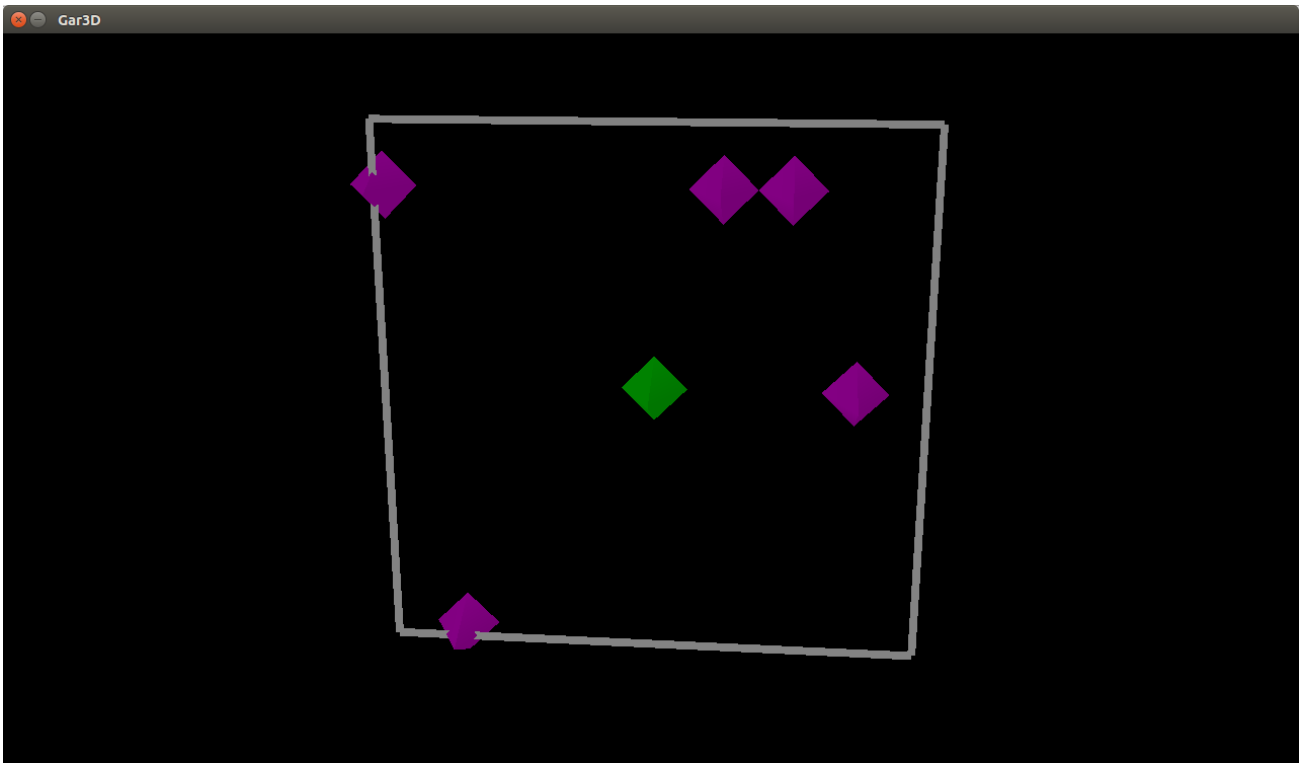
After 5 time steps, the enemy grow to full strength and instead you die if you walk on top of the enemy. Add a victory condition if you manage to kill all the enemies. (This should be easy if the only enemy is at (0.5, 0.5).)

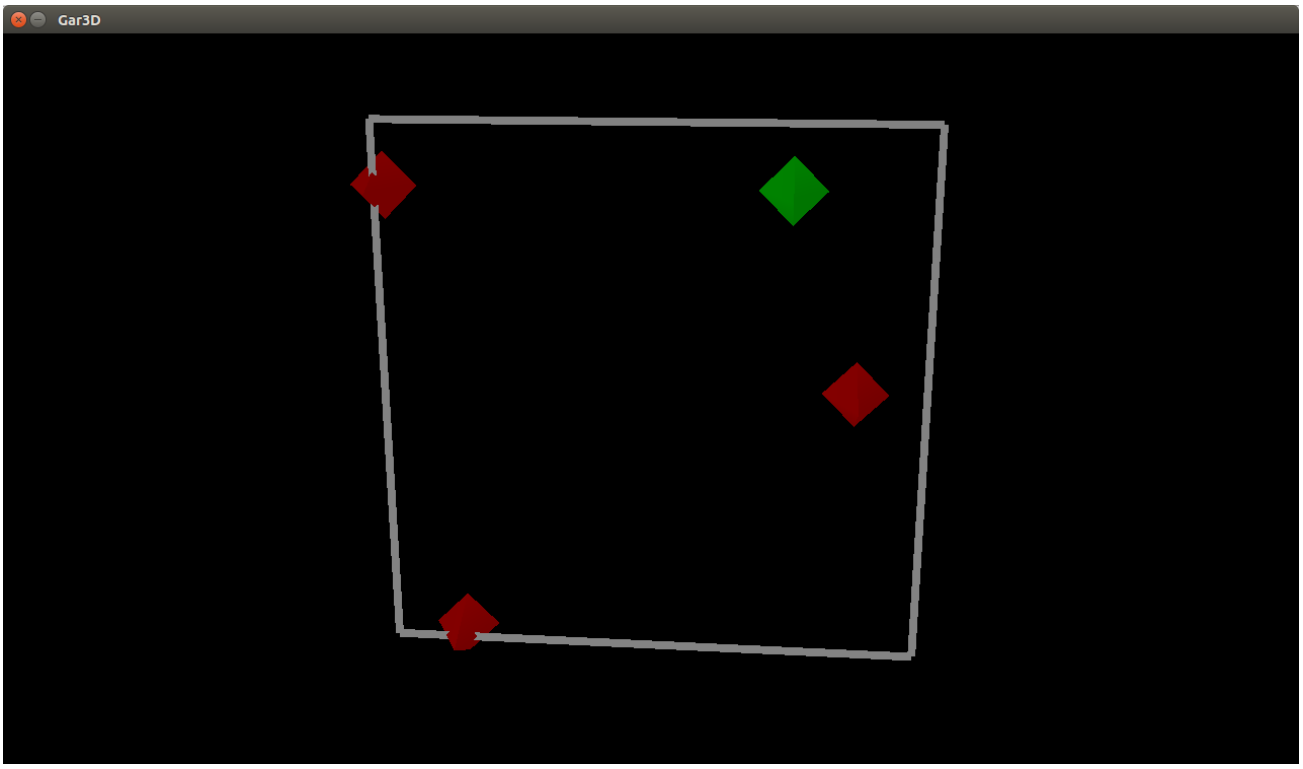




2) Multiple enemies

Rather than having a single enemy at (0.5, 0.5), make 5 enemies spread randomly throughout the board. Put one of them on the 9x9 grid. This means their x-position and y-position need to be either -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5 or 2.

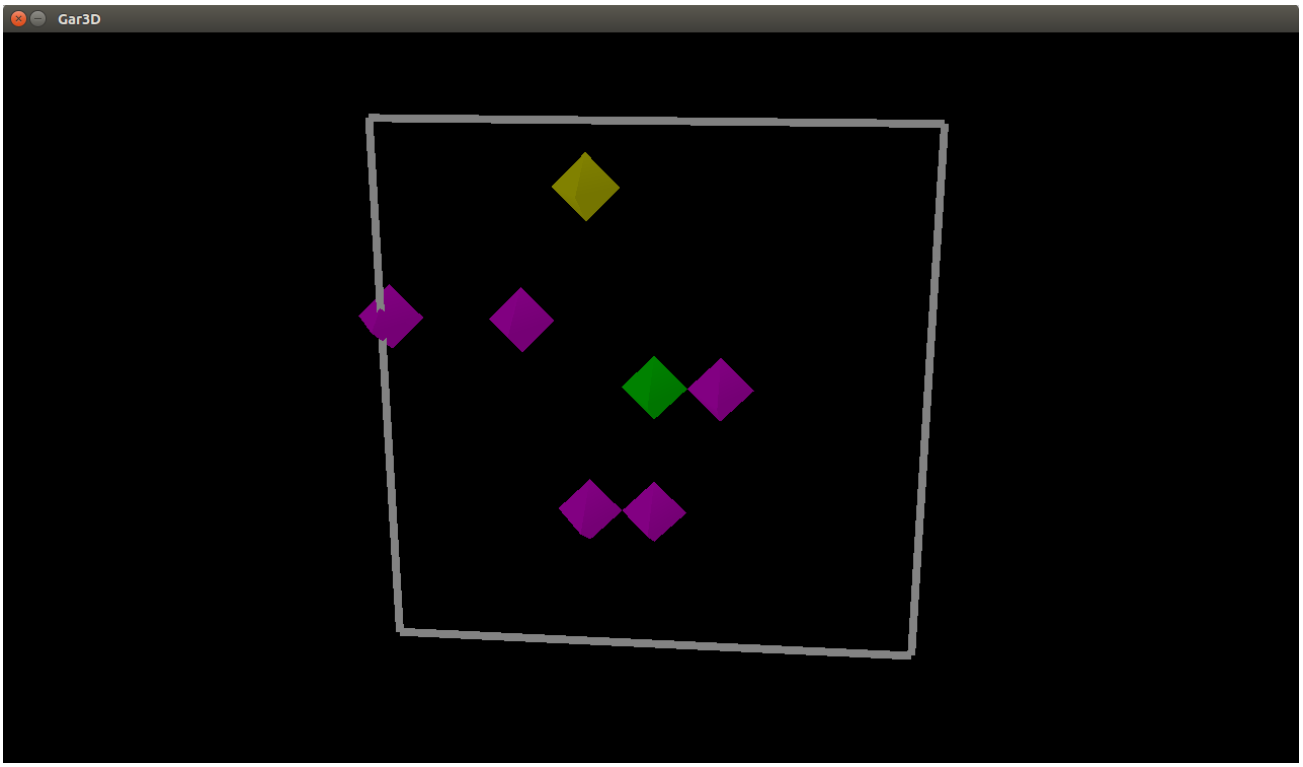

Gar3D

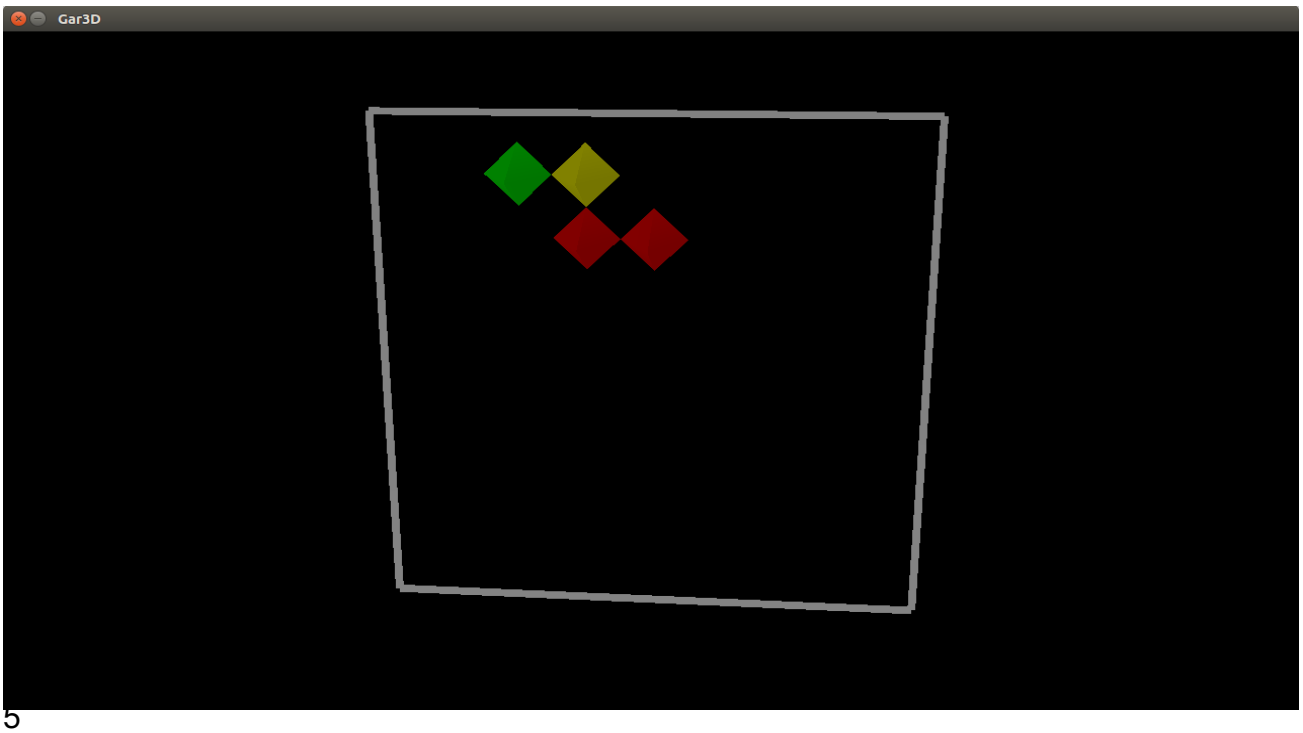

Gar3D

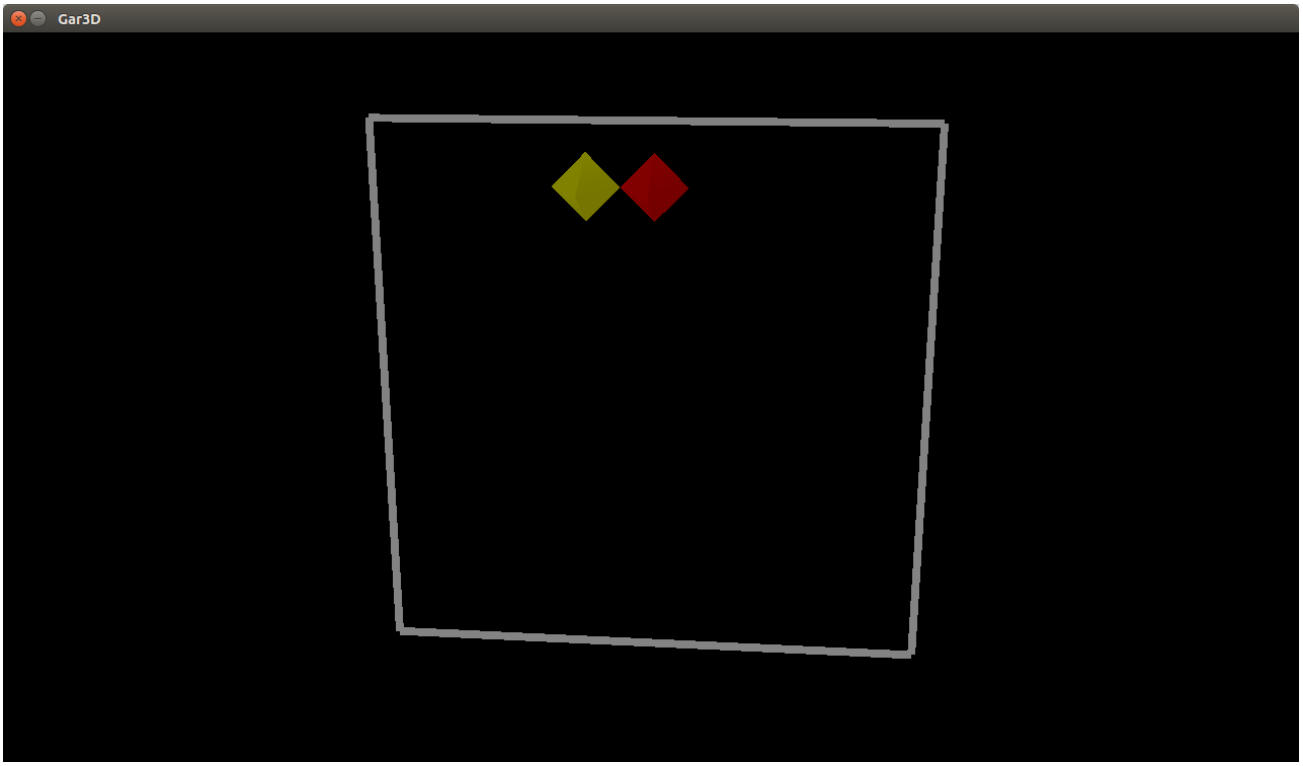
3) Power-ups

Add a power-up to the board. This should be another class, that you can inherit from the moving object class. Have one power-up at the start of the game placed randomly on the board. If the player walks on top (or near) the power-up, the player should become super-powerful for 3 moves. In this super-powerful state, the player defeats fully grown enemies rather than losing to them.

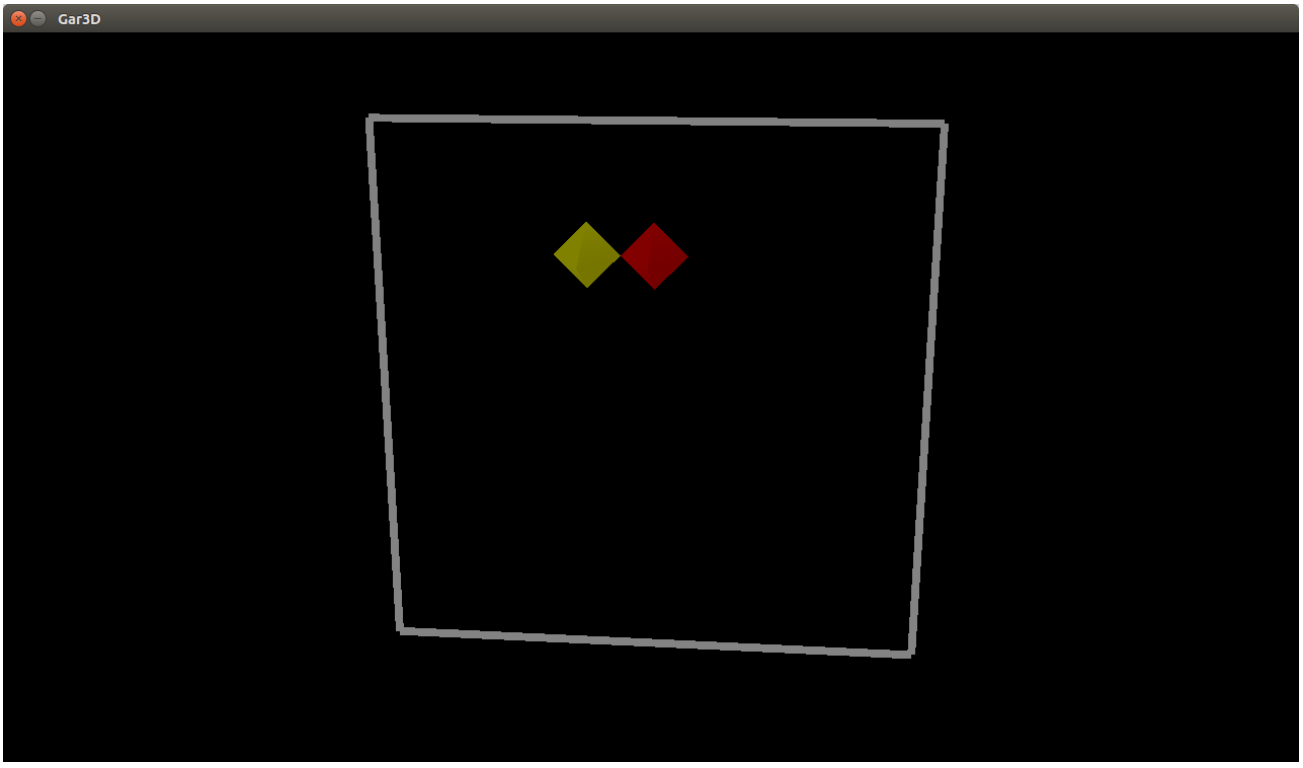
Have the power-ups be a new color and change the color of the player when this power-up is in effect.







(now that have power-up, can move with it)



Challenge

1) Continuous game
Modify the game so that the power-up does not start in the game. Instead, there is a 10% chance that the power-up spawns each movement (1% for continuous movement). Also have a 20% chance to spawn a new enemy at a random square each movement (2% chance for continuous movement).

Do not spawn monsters on top of the player, as that would be mean... (It is okay if this slightly reduces the chance that monster's spawn.)

2) Powerful monsters
Make a new monster type. This monster starts are full strength and still kills the player even when the player has a power-up. However, make this monster die after moving 4 to 6 times (randomly).

3) Cloning power-ups
Make another power-up which when stepped on makes another copy of the player randomly on the board (not occupied by a monster). Both copies of the player move in the direction indicated by the key pressed. You can choose how the monsters try to move towards the player and it's clone. However, even if the “original” player dies, the game still continuous until all copies of the clone are dead.

Hint: Rather than separating variables between “Enemy” and “Player”, the “Game” class can have just a single array (of pointers) that keeps all the enemies, players, clones and power-ups in the same spot/variable.

Appendix

In the “Gar3D” class:

drawLine(x0, y0, z0, x1, y1, z1)	Draws a line from (x0, y0, z0) to (x1, y1, z1).
drawPrimitive(primitive, orientation, posx, posy, posz, sizex, sizey, sizez)	Draws a primitive object. Available primitives and orientations are shown in Gar3Dclass.hpp.
drawPrimitive(primitive, orientation, posx, posy, posz, size)	Same as above, but scales by a single size
renderFrame()	Renders drawn objects and updates the input handle. If this function is not called each frame, the program will not respond.
shouldQuit()	Returns true when the program should exit. This occurs when the user presses the escape key.
color(r, g, b)	Sets the color of the objects drawn. May be changed for each object. r, g and b are between 0 and 1 for “red”, “green” and “blue”.
input()	Gets the input handle. Methods for the input handle are described below in “InputHandle”.
lineSize(size)	Sets the thickness of lines .
backgroundColor(r, g, b) Sets the background	Sets the background color of the window. Value

color of the window. Value is only used when renderFrame() is called.	is only used when renderFrame() is called.
backgroundColor(color)	Same as above, but with a color type rather than 3 int types
deltaTime()	Return the amount of time (in seconds) elapsed during the previous frame (last renderFrame() call).

In the “InputHandle” class :

buttonPressed(Key)	Returns true on the first frame the button is pressed .
buttonHeld(Key)	Return true on every frame the button is held down .
buttonReleased(Key)	Returns true on the frame the key was released .
mousePosition(x, y)	Changes x and y variables to the mouse cursor's x and y position in the window.