Code Report

# Gameplay/ Design

My initial idea for the game was to recreate the classic Snake game, which I started out with but later modified to be another simple arcade game which I titled Run Piggy.

The aim of the game is to collect as many coins as possible without colliding with any of the spheres running across the grid.

# Classes Used

**Pig.cpp-** This class contains all the functions and variables used for my main character and user. I had standard functions for movement like forward, backward, left and right, which I kept in public along with variables that I used to access other private functions and variables. I created a unique pointer m\_pig for the pig in my NGLDraw header file because it made it easy to work with all the elements in this class.

I set the direction of the pig to (0,0,-1) by default so that it would move upwards once the game begins. Setting the equations m\_dir\*=m\_speed and m\_pos+=m\_dir under my move() function and updating move() every frame meant that the pig would be moving constantly. I thought that would make the game a little more difficult.

\*I initially wanted the pig to be able to stop moving when it reached the edge of the grid but I ran into some issues with that because I couldn't get it to continue moving after one of the direction controls was pressed, so I used a set of if loops to ensure that when the pig reaches an edge on the grid, it would re-appear on the opposite edge.

**Coin.cpp-** This class has a similar setup to my Pig class, but I included a spawnCoin() function that changes the position of the coin to a random point on the grid, whenever the pig collides with it(coin). I used this equation to spawn the coin to points within these values.

m\_coinPos.m\_x = std::rand() % 50 +(-25) +0;

m\_coinPos.m\_z = std::rand() % 50 +(-25) +0;

**Bullet.cpp-** I wanted 3 sets of spheres running across the grid, in different directions. To make it easier to control each set, I created instances for each individual sphere in my NGLDraw header file. This made my collision detection function a large collection of if loops and although it was quite tideous and clumsy, it helped me control each individual bullet.

Because I wanted the spheres to go in different directions, I used 2 boolean function to flag the set of spheres that were coming in from the right and those that were coming in from the top. That way I could reverse the direction of just those spheres if and only if the functions were true, without altering the direction of the first set of bullets that were coming in from the left. I also wanted all the spheres to sort of bounce off the edges of the grid so I adopted 2 inline void 'reverse' functions (for m\_dirX and m\_dirZ) that just returned the negative of the direction of the bullet(s):

inline void reverseX(){ m\_bulletDirX\*=-1;}

inline void reverseZ(){ m\_bulletDirZ\*=-1;}

**Lives.cpp-** I created instances for the lives just like I did for the bullets and I included <ngl/Obj.h>

in the header file so that I could import a mesh for the lives.

**Gameover.cpp and StartScreen.cpp-** These two functions are identical apart from their variable and function names. I imported meshes for both classes just like I did for the Lives, Pig and Coin classes. I encountered many issues when trying to get the startup mesh to draw the the texture properly. My final startup screen was quite shabby as a result and has plenty of room for improvement.

\*The boundary() function that I used to limit the pig's motion was only half-functional, as only two out of the four if loops were working, although they were pretty much identical but for a few variables.