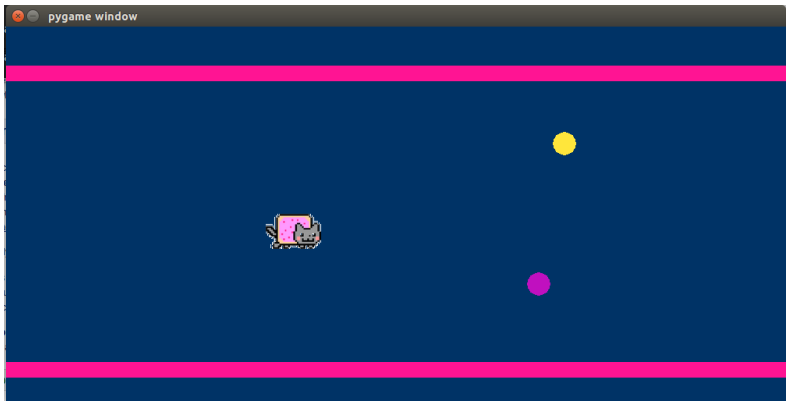
Aditi Joshi and Jessica Sutantio

Project Overview:

We created a pygame game that is based on the game "One More Line". In this game, the objective is to to forward on the screen while avoiding the circles. In order to avoid the circles, you have to click and hold the mouse button and it will take the player around the circle. He or she then has to let go of the mouse so that the player does not run into the wall or another circle. Our game is pretty similar. The player is now represented as Nyan Cat. The cat has to avoid the circles and the walls just as in our inspiration game. However, there is now the added challenge that you have to keep the cat character on the screen and the cat now moves in a spiral motion away from the circle instead of a circular motion.

Results:

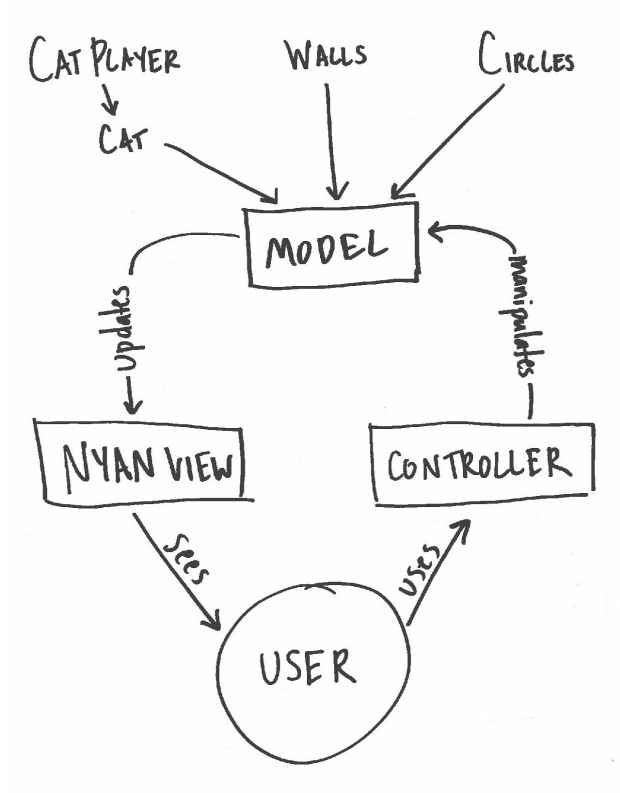
Here's an image of someone as if they would be playing our game in real time.



Implementation:

Our UML class diagram is located lower down in this section. Basically, our structure consisted of a model view controller system. The model consisted of three parts: the cat, the walls, and the circles. The cat also had a "CatPlayer" class which was basically the representation of the cat at a certain point in time (it set the position of the player as it moved in the game play). The user uses the controller (which is just the mouse click function) and the that changes the model which in turn changes the walls, circles, and cat. These changes then update the Nyan View class, which displays these three elements to the user, who then sees Nyan View.

One decision decision that we had to make was how to split up the update functions. Rather than putting all of the updates inside of a model update function, we created three different update functions for the cat, walls, and circles which we then called inside of the model's larger update function. This made the code a lot cleaner and easier to read because we knew that the different things that needed to be updated were doing so within the larger function. It also let us call out specific updates in different modes (for example, when the mouse was clicked, only the cat needed to be updated, so we didn't need to call the model's update function but could just call the cat's update function).



Reflection:

One big problem that I think we faced was that we started with a completely different project at the beginning and then realized that we could not complete it because of API limitations. Basically mid-way through our project, we decided to change to a game project rather than a data visualization one. If we would have started with this idea at the beginning, it probably would have been in the scope, but because we started mid-way through the allotted time, it was more difficult than we imagined. Coding-wise, one major problem that we faced was the naming of the variables. It got confusing as to which variable was mapping to what object. It took some time getting used to the variable names that we used (many of which were long), which makes our code less would write a function readable. We tested each part of our code in increments. We would write a method or a class and then run through the code, making sure that everything worked like we wanted it to. We would print out important variables when things went wrong to see what exactly was going wrong.

We had planned to split up most of the work with pair programming and I think that was the right way to go. Most of the time we did pair programming, which worked extremely well. The times that we split up the work, a lot of our time was discussing what we did individually when we got back together. Also, in a lot of the split up work, we had many more github problems that we did not face when we were pair programming. The only real issue we had when working together was that we sometimes did not clearly document what we were working on individually when we did work individually, so it was more difficult to catch each other up when we did come back together.