Project Writeup:

Cap’n Chicken

Python Game

Daniel Daugherty
Kevin Zhang

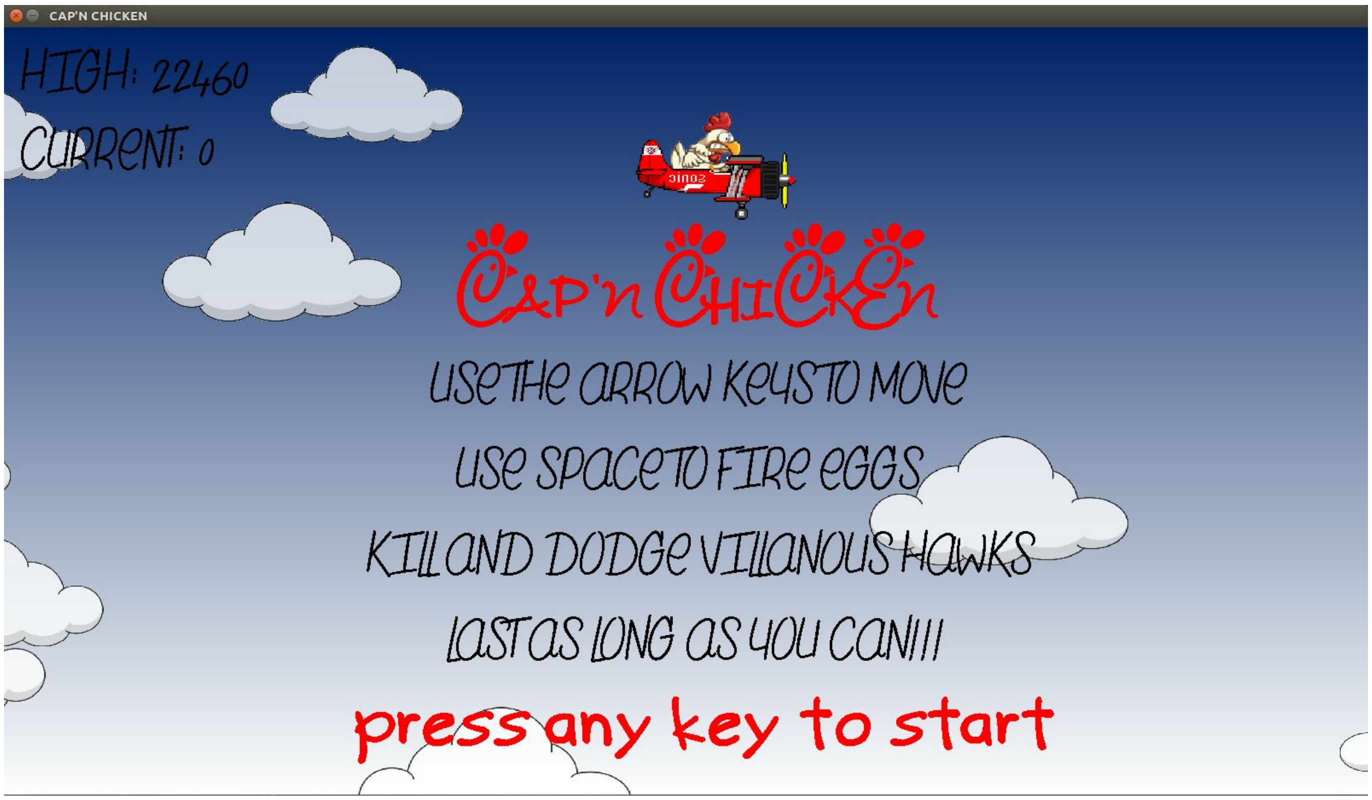
Software Design 2016

Project Overview:

Wait what…? That's right. Cap'n Chicken is gonna do it. Finally after several years of being contained in Farmer Billy's chicken coup, Cap'n Chicken has worked up the guts to steal Billy's plane. Help Cap'n Chicken in his rebellious skydive to the Mountains of Freedom! But beware, the journey to the Freedom Mountains is not an easy one… Watch out for predatory hawks and the occasional Alpha… Good luck and have an eggcelent dive!

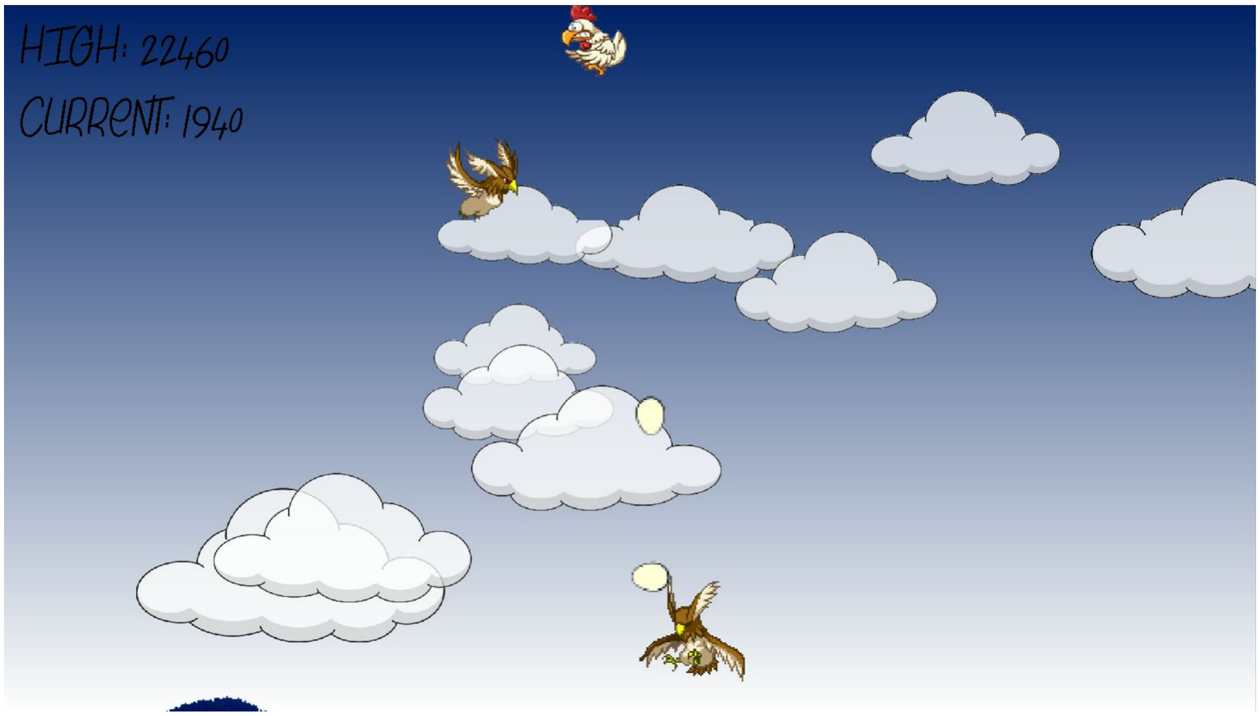
Results:

We were able to create a vertical scrolling shooting game in which the player controls a skydiving chicken who must dodge and shoot predatory hawks that seek the chicken. The chicken has the ability to move quickly through the air and can shoot eggs downwards to kill hawk. If touched by a hawk, the chicken dies and the game is over.

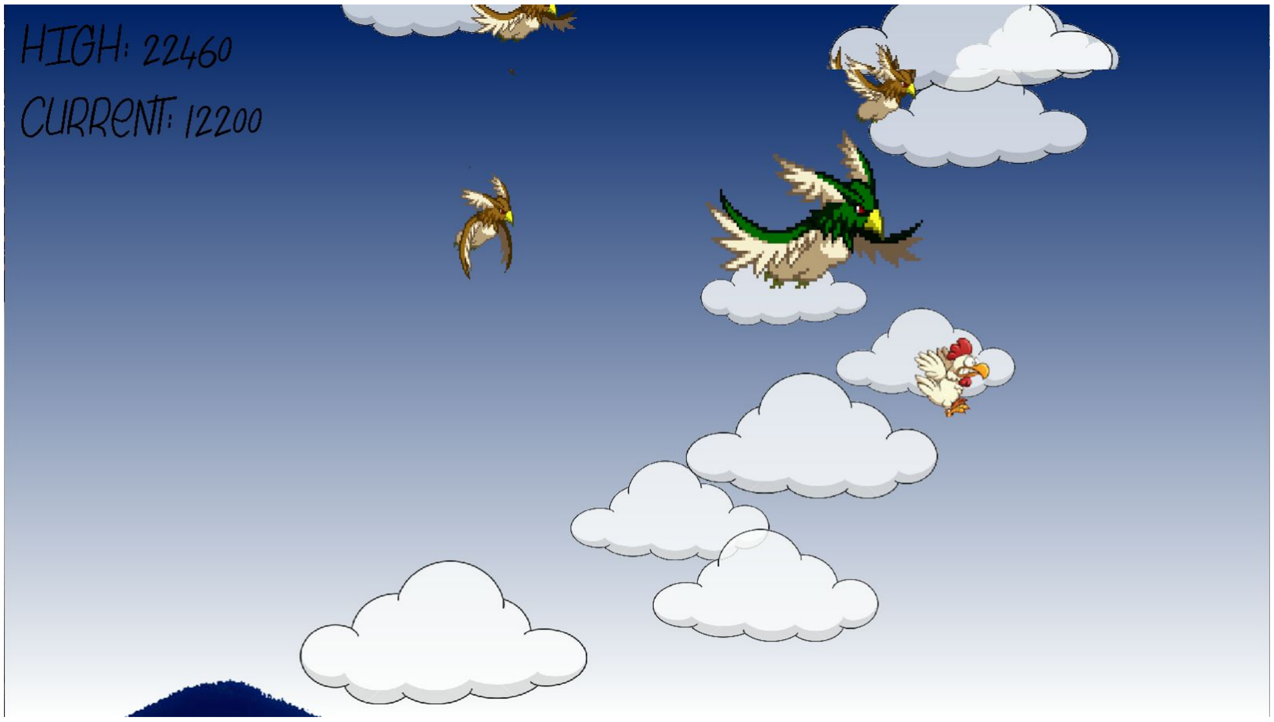


Start Screen

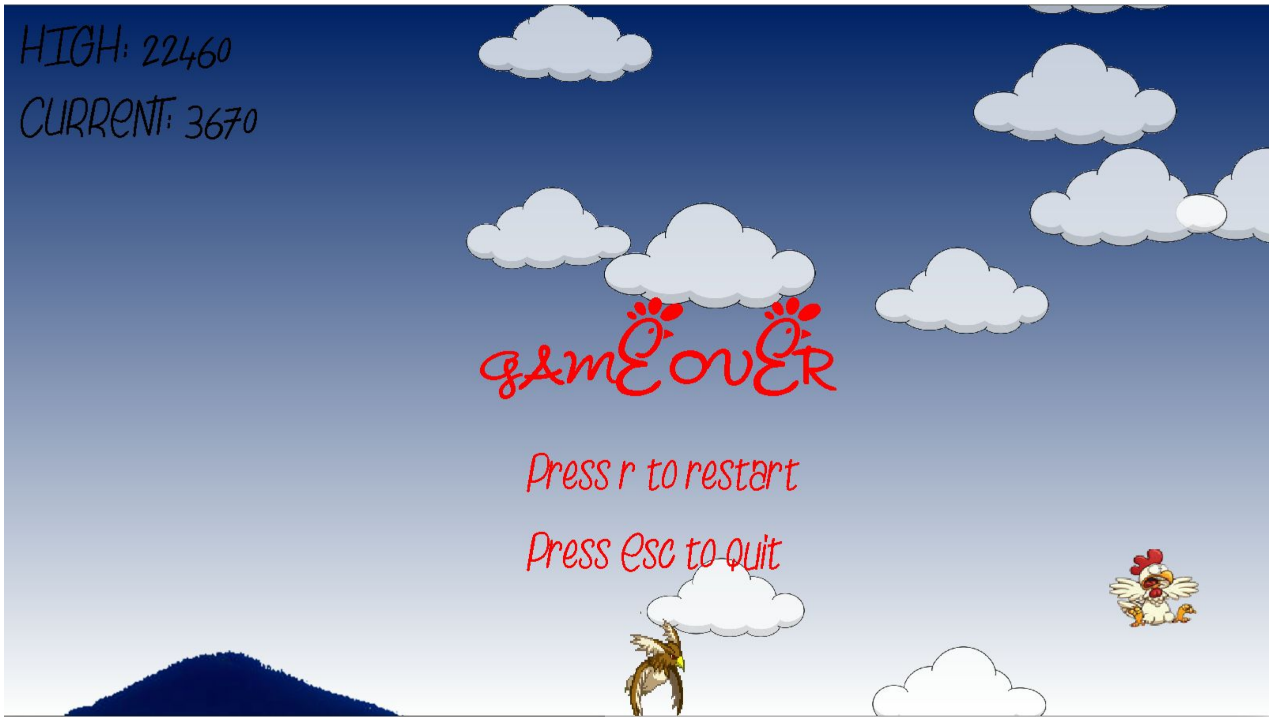
The object of the game is to collect as many points as possible and beat your high score. The game starts with two hawks seeking out the chicken and as the player's score increases, the swarm of hawks gets bigger. After the player has reached a score of 10000, an Alpha hawk will start to seek out the chicken. Killing regular hawks earns the player 1000 points and killing an Alpha hawks earns the player 5000 points. We believe there is a good game difficulty balance in which killing hawks yields a lot of points, however, it is risky to travel to the top of the screen to launch eggs because the player has less reaction time to avoid incoming hawks.



Dropping Eggs on Regular Hawks

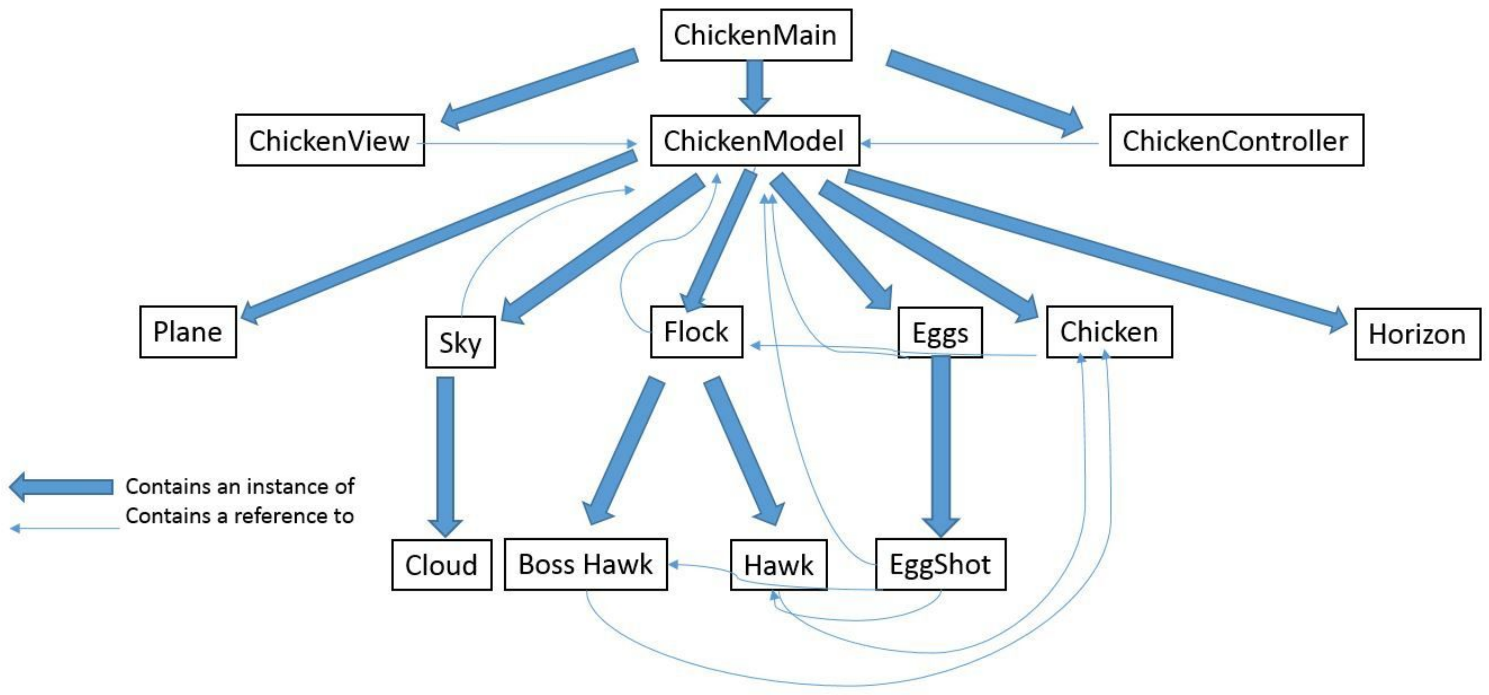


Alpha Hawk



Game Over Screen

Implementation:



We created our game using the MVC code structure. The ChickenModel class utilizes the ChickenController and ChickenView class to move the chicken around and display the graphics on the screen. Our game consists of several classes for each element within the game and we created classes for groups of elements within our game. For example, instead of creating a ton of hawks within our model, we create a flock which is a list of several hawk objects. This creates a sort of hierarchical structure in which the classes make up the groups which make up the model. Our game runs on a loop that is based on pygame's built in clock. This allows us to continually update the view screen and control the chicken.

The biggest design decision we had during this project was balancing the game difficulty. One option we had was programming the hawks so that they start off moving in straight lines and then progressively get smarter over time so that they start tracking the chicken. We thought that this would allow for a lighter difficulty curve for the game so that it starts off easier. We decided against this idea because the moment the hawks get smarter and start tracking the chicken, the difficulty increases greatly. Instead of a gradual difficulty increase, the difficulty spikes when the hawks start tracking. To get a more gradual difficulty increase, we decided to add more hawks as the players score gets higher.

Reflection:

We are very happy with the progress we made throughout this project! Both of us have not had any previous experience working with pygame or video game design so we feel like we learned a lot. We do believe that our project was appropriately scoped because our final code was able to incorporate all of the elements we talked about at the beginning of the project which included, vertical scrolling, hitbox collision, score accumulation, and animated characters. One of the biggest struggles throughout the project was figuring out where to incorporate certain changes within the MVC structure. For example, to get the start animation working, the use of deep attribute changes had to be implemented to change the direction of the clouds. In order for the clouds to move during the start animation, the model needed to run with the clouds already instantiated. To change the direction of the clouds, the cloud's velocity had to change from the top level of the code.

We found throughout the project that we were most productive while working side by side. Our team process consisted of about half pair-programming, and half merging code via Github all outside of class. We scheduled 2-3 hour meetings about 8 times over the scope of the project. One of the biggest problems we found working together was merge conflicts on Github. We would work on different aspect of the game at the same time on our individual computers and then try to merge our code after spending 2 hours making changes which would then take another hour to resolve. After struggling with this several times, we decided to switch to more pair-programming and working on smaller changes before trying to merge. In future partner programming projects, we think that pushing code much more frequently would really help with reducing wasted time fixing merge conflicts. We were also thinking about possibly branching the code on Github so that we would have to worry about as many merge conflicts when pushing code. We had a great experience with this project and are really happy with our final game!