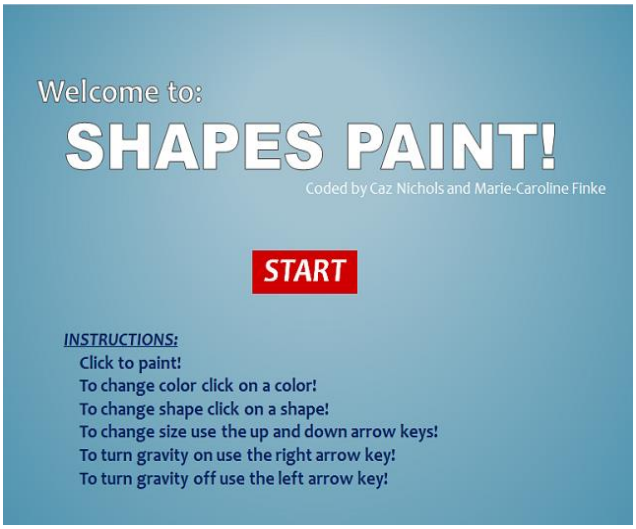
ShapesPaint interactive paint physics program
Marie-Caroline Finke and Caz Nichols

Project Overview
ShapesPaint is a sandbox game in which the user can select the color, shape, and size of a brush, and can turn on gravity to watch the painted shapes fall down.

Results
Our game starts with a screen where the user can read instructions on how to use the controls, and press the START button to begin playing. The game allows the user to select colors by clicking on a colored square, select size using the up and down arrow keys, and turn gravity on with the right arrow key or off with the left arrow key. Gravity affects shapes proportionally to their width. The user can draw while gravity is turned on or off.

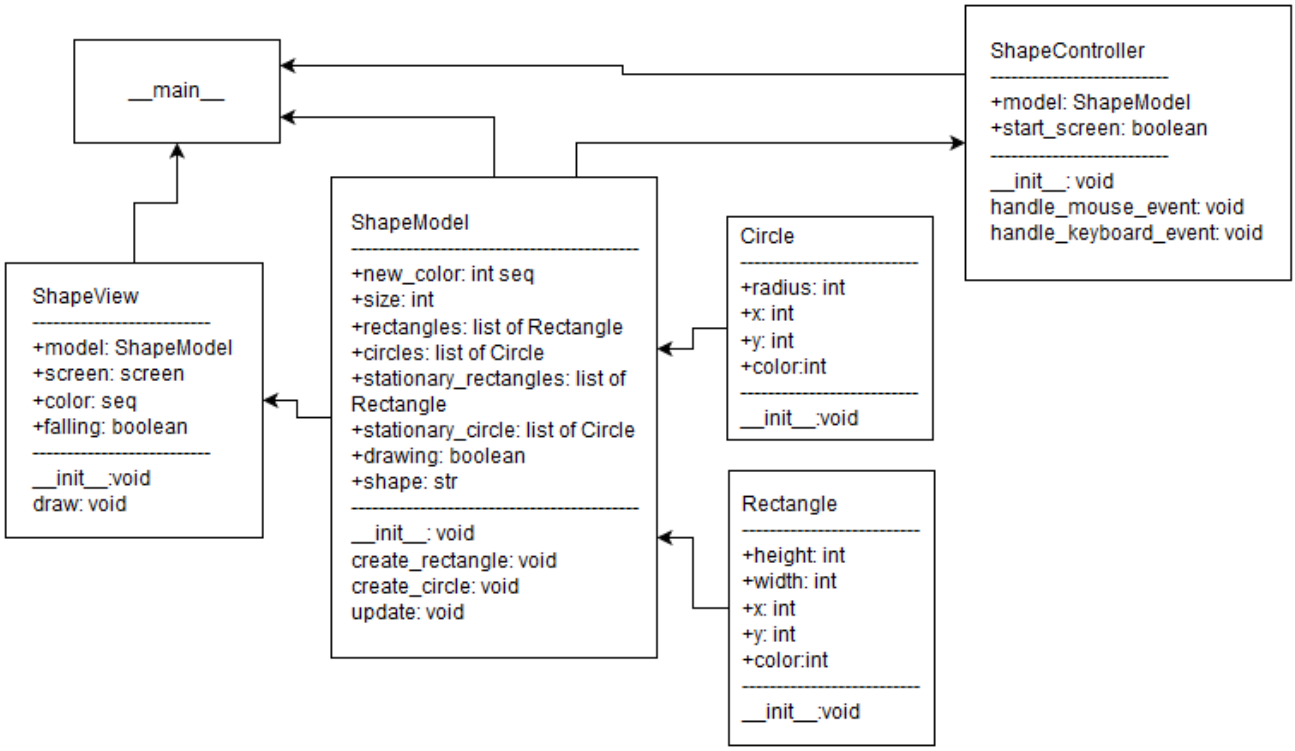


starting screen and screenshot of the paint program in action

We used the model-view-controller architecture. The model stores rectangles and circles; the view keeps track of their positions and updates the graphics accordingly; the controller gets user input and changes brush/graphics options.

Implementation
ShapeModel stores information about Rectangle and Circle objects and its update function creates new shapes when called by the run loop in __main__. ShapeView takes a ShapeModel object and in the draw function it cycles through the lists of shapes to render shapes on the screen when called by the run loop in __main__. ShapeController takes mouse position and button input from __main__ in handle_mouse_event and key input in handle_keyboard_event to change information in a ShapeModel object.

To change the shape and color handle_mouse_event detects the mouse position when clicking, and if it is in a specific range it changes the appropriate variable in a ShapeModel object. The start screen is similarly implemented. We used lists of shapes (Rectangle and Circle objects) to which new shapes could be appended when they were created and from which shapes that had fallen off the screen could be removed. This also made it easy to use for loops to cycle through each element of the lists and draw the shapes.
We chose to use an image file as our starting screen rather than attempting to code it in pygame because the layering of text and rectangle elements was suboptimal. Then all we needed to do was blit the picture to our screen, which was much less trouble than compiling a collection of objects.



Reflection
Our project was not only inappropriately scoped, but not actually scoped at all. We were advised to create a much more complex physics simulator that would cut out shapes and have them bounce off each other, which is next to impossible with our level of skill. We simplified the project, but were not in time to make a proposal that reflected achievable goals. We unit tested along the way, haphazardly and without direction. However we did succeed in making sure everything worked and fit together, and our code is fairly solid and readable.

Tis project was extremely useful from a learning standpoint because without being forced to figured out

how classes and the model-view-controller architecture work, we never would have been able to understand it as intuitively as we do. It would have helped to know beforehand about the limitations of pygame and the code flow.

We enjoyed pair programming and found it extremely useful to do it together, switching off the driver and navigator. When we coded components of the program separately they fit together surprisingly well and there were few problems integrating our code. I think we were on the same page as far as how the program works and we did pretty equal amounts of work and learning. This is especially impressive because Marie-Caroline was bed-ridden and delirious for about half of the project time.

Next time we would ask for more input from NINJAs and teachers about how feasible the project is, because we went in without any idea whether our program would work or how.