Andrew Pan + David Papp
SoftDes

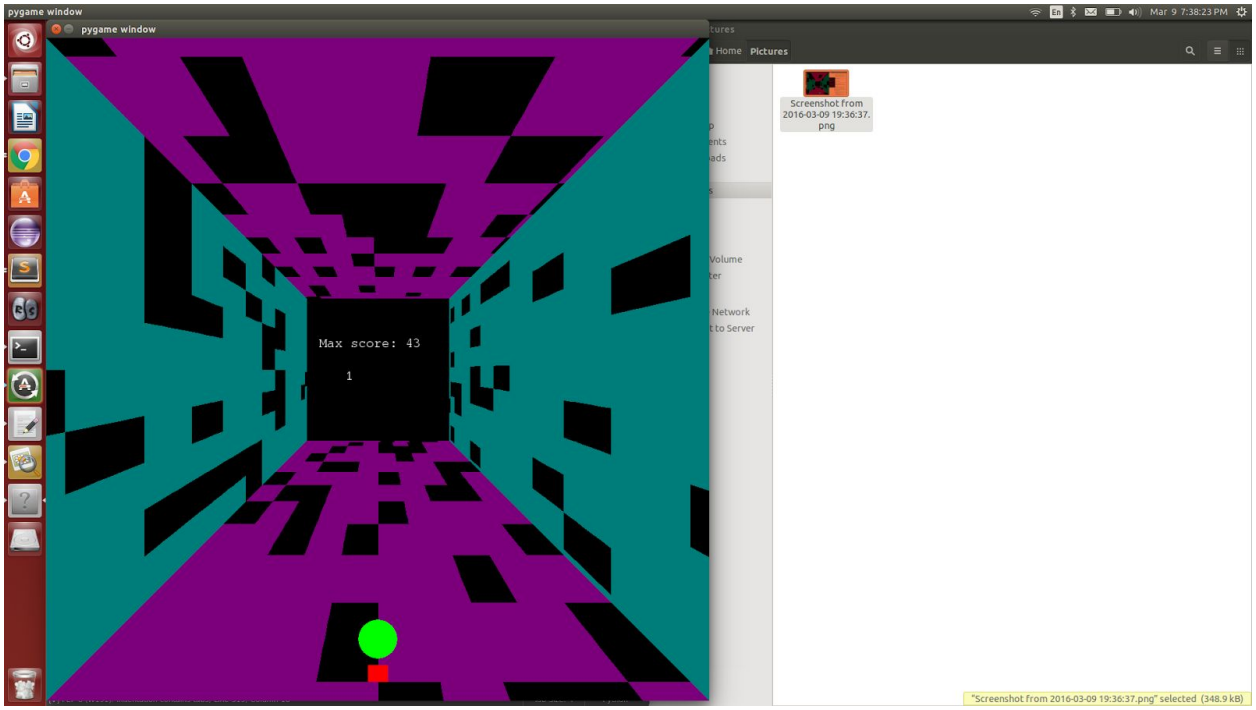
Run

Project Overview:

Our goal was to mimic the popular 3D game run, where a player is running in the z-axis and must stay on the platforms. Since we both have had previous experience with many objects and classes, we were instead hoping to learn about 3D graphics, OpenCV, multi-threading, and pair-programming.

Results:

The first stage of our project involved implementing the game and controlling it via keyboard. Much like the original game, the player is put onto a plane that moves in the z-axis. The plane has gaps that the player must avoid by either jumping over it or going around it. What makes the game interesting is that there are four planes: a bottom, a top, a left, and a right, and each plane has randomly assigned gaps. The player can move onto another plane by moving sufficiently to one side, and this rotates all the planes. If the player falls into a gap, the scoreboard resets to 0.



One of the challenges was mimicking realistic 3D grid graphics. We accomplished this by creating multiple lists of vectors in slope intercept form and finding their intersections, and using these points to map out polygons via pygame. The end result was that the 3D grid we created was fairly realistic. We also made the game fairly customizable: you can choose the frame rate, size of the board, and the frequency of gaps in the planes.

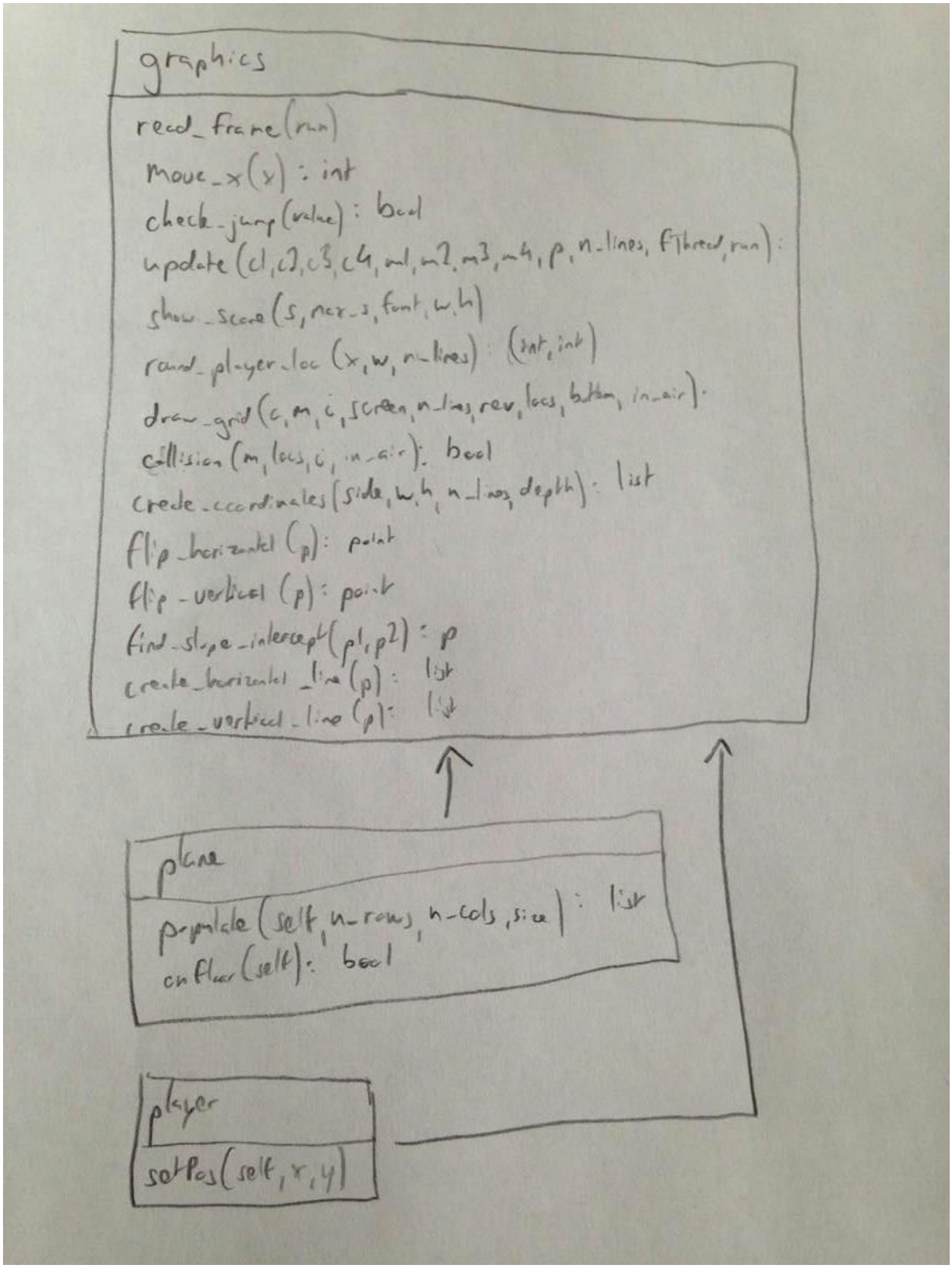
The next stage of the project was to add OpenCV as a control option to the project. OpenCV's built in algorithms detect the player's face and move the character according to the player's

head movements. For example, tilting your head to the left will make the character go left. We quickly found that OpenCV's frame capture and face-detection is extremely slow and thus it slowed our game down to about 4-6 fps. To solve this, we implemented multi-threading. This solved the issue with the map's frame rate, but OpenCV's frame capture was still very slow, so while the map would move at a normal frame rate, the player's input would only register a few times per second, rendering the game unplayable. Regardless, we learned a great deal about multi-threading.

Implementation:

Our project focused primarily upon a game that gives the player a 3D perspective of sorts, which caused a fair portion of our work to go into creating the graphics of the game. We ended up having all of the calculations for player controls and physics take place in the same file as graphics, and only had two real classes: a player character and a class to define the planes the player can walk on. We had another class, but it only existed for the purpose of determining when to exit a thread. This was fine because we did not need to instantiate multiple objects in our game. All calculations were handled in individual functions throughout our graphics file, as well as the facial controls.

The coordinate system to create the 3D effect was defined prior to any actual drawings, and was then referenced with a matrix of points to draw the floors and walls with holes in them. The player can be moved through key inputs or the location of a face read in using OpenCV, which caused some problems for us initially. As previously mentioned, using OpenCV in conjunction with our game caused the game to be extremely choppy, as it had to wait for OpenCV to properly read in the new set of facial coordinates in order to load the next frame. To solve this problem, we chose to run both processes in separate threads, which made the game reasonable but very difficult to control with the facial position input, which we determined to be a reasonable compromise as the game could still be controlled with keyboard inputs in this state.


graphics
read_frame(run)
move_x(x) : int
check_jump(value) : bool
update(c1,c2,c3,c4,m1,m2,m3,m4,p,n_lines,fThread,run):
show_score(s,scr_s,font,w,h)
rand_player_loc(x,w,n_lines) : (int,int)
draw_grid(c,m,i,screen,n_lns,rev,locs,button,in_air):
collision(m,locs,i,in_air): bool
create_coordinates(side,w,h,n_lines,depth) : list
flip_horizontal(p): point
flip_vertical(p): point
find_slope_intercept(p1,p2) : p
create_horizontal_line(p) : list
create_vertical_line(p): list
plane
populate(self,n_rows,n_cols,size) : list
onFloor(self): bool
player
setPos(self,x,y)

Reflection:

We were perhaps too confident in our ability to make this game, and this resulted in careless programming and testing. We did not use unit tests, we rarely commented our code, and we did not include any docstrings until we had already finished writing our code. Our program was not divided up into manageable and understandable parts and we did not take a particularly objective oriented approach. We only had two classes. Our reasoning behind this was that since our game doesn't involve many different pieces, we would never need to make multiple instances of some object and thus it isn't really necessary. We instead mainly focused on the 3D graphics, OpenCV, and multi-threading part.

Regardless of our mistakes, we are satisfied with our progress and especially our division of labor. We had a good understanding of how the game would function as a whole, which is why we had little need for docstrings or comments. Often our interactions would be as simple as "Andrew give me an n by n matrix and I'll draw it on the screen," or "I'll make a character class that moves according to the keyboard." Going forward, we will need to structure our code better, but we believe this will be no problem. Overall, we feel good about what we accomplished.