3D Scrabble

David M. Stone

December 9, 2016

Abstract

The goal of this project is to simulate a three-dimensional Scrabble game with Python and OpenGL. Two players compete on the same machine on a cubic adaptation of the classic board layout. Players require skills of vocabulary, strategy, and anagramming, though there is an aspect of luck in drawing tiles. Scrabble is a board game staple, and a three-dimensional variant provides new challenges and opportunities for an avid player.

1 Acknowledgements

I, the author, would like to thank the following: Professor George Francis, for teaching and guidance; Class Mentors Molly Fane, Sasha Lamtyugina, and Nathan Walters for advice and consulting; and Cynthia Capota and Daniel Zou for their review and recommendations. Their support is the reason this ambitious project achieved as much as it did. Furthermore, I would like to recognize the following for their code, ideas, or other helpful contribution to my project: Robert Kaufman, Stan Blank, Alex Bourd, Matthew Stiak, Jeff Knupp, Mike C. Fletcher, www.opengl.org, wiki.python.org, pyopengl.sourceforge.net, nehe.gamedev.net, forum.lwjgl.org, and stackoverflow.com.

2 Background

Scrabble is a two-dimensional board game in which players take turns to form words and place them on the 15x15 board grid. The letters which form the word must be connected linearly, and the letters must be chosen from the set of tiles already on the board and tiles in the player's rack. There are 100 tiles, each bearing a letter, the distribution of which was calculated by creator Alfred Mosher Butts [1] through analysis of letter frequency in prevalent publications (e.g. The New York Times). Each letter has a point value, and the sum of the points of each letter used is the score for the turn. In crossword style, each word (2-15 letters) formed must be permissible, as defined by a preset dictionary. A player's turn consists of: choosing a play, placing tiles, finding the score, and then refilling the rack up to seven tiles. A player may, instead of making a play, trade in any number of tiles from the rack back into the bag and draw new tiles. [2]

2.1 Special Rules

Of the hundred tiles, two are blank (devoid of letter or point); these may be used as any letter, but once put on the board, represent that letter for the remainder of the game, and count for no points. Bonus point spaces are dispersed across the board, and are valued multiplicatively. Finally, if a player successfully uses all seven letters from the rack, that is sometimes called a Bingo and is worth an additional 50 points. [2]

3 RTICA

The board is a three-dimensional rectangular lattice, and a set of letter keys (QWEASD) are used to direct a Selector across the board. The seven letters of a player's rack are displayed across the bottom of the screen; when a player enters a number between 1 and 7, inclusive, that tile (counted from the left) will be put into the location of the Selector. The view of the board is initially orthographic, in order for the player to see the most of the board, with the availability to spin it through the arrow keys in order to access different angles. The special value locations are marked by low-alpha shaded cubes with red representing double word score and blue representing double letter score. Once a player is done with the turn, pressing Enter will submit the play and transfer to the next player. Furthermore: pressing Escape will quit the game, p will display the overall points, v will toggle the volume of the tiles, and h will display a help page with all of the keystroke information. Here is the help page:

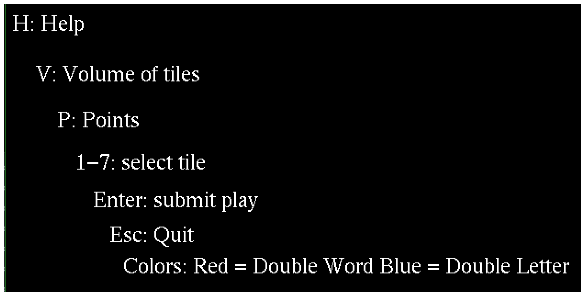

H: Help
V: Volume of tiles
P: Points
1−7: select tile
Enter: submit play
Esc: Quit
Colors: Red = Double Word Blue = Double Letter

Figure 1: This is the help page's information against a black background.

4 Implementation

4.1 Development

For my Seminar, I had a visual 2x2x2 board, the ability to rotate the view, and a representation of a tile. However, the board was made simply of lines and the tile representation was nothing more than a green-filled quadrilateral. I desired objects that could interact with each other, to simplify the expansion of the board from 2x2x2 to 9x9x9, to help assign tile locations, and to determine location selection by players. To this end, I wrote and debugged a Box class, with methods to draw itself about its center and shade itself (to become a Special Point space). Next, I focused on creating adequate tiles. It was recommended by Professor Francis I make three-dimensional, extruded letters, so I researced FreeType and FTGL libraries which promised to have the functionality I required. However, documentation was thin and examples were sparse, and what help existed was in C. After extensively modifying the C examples over Fall Break but achieving no progress toward a Python script, I gave up and resorted to my second option: textures. After working with two mentors, Molly Fane and Sasha Lamtyugina, we found PILLOW and its child Image, which opens a picture file and converts it to a texture. Furthermore, Sasha discovered that the code must specify which vertex of the quadrilateral corresponds with which corner of the image. I made a Selector, a white box whose position changes in response to keystrokes, which replaced the mouse as the method to select which location on the board to place a tile. Once I had included keystroke responses for a help page, score display, tile volume, and ending a turn, I hard coded the boxes and Special Point spaces for the 9x9x9 board. Finally, expanding the board made the size slightly too large for the screen, and the interlacing lines were confusing. To address these problems, I made the lines grey instead of white, and I changed the boxes (and tiles) to have side lengths of 1 instead of 2.

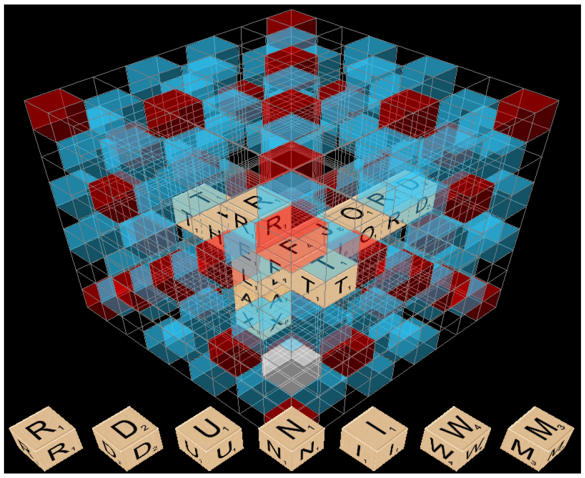
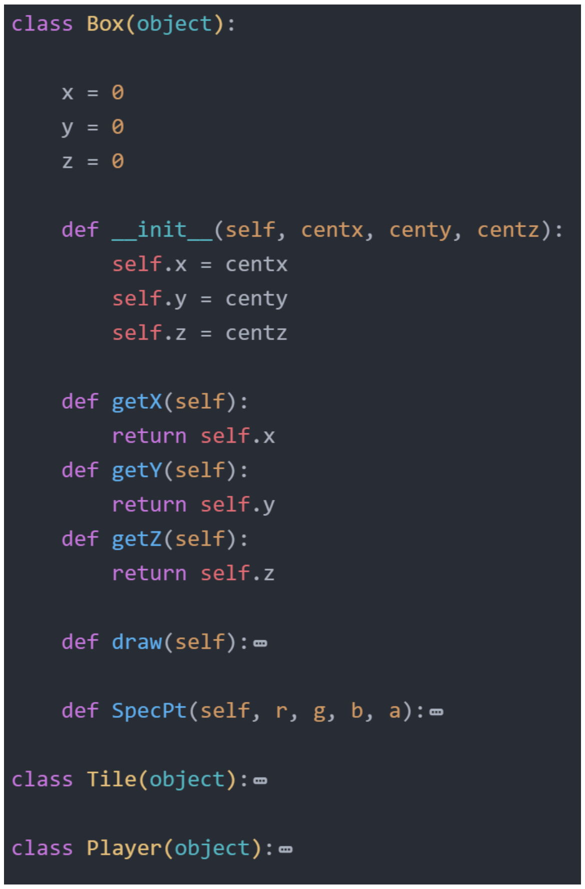


Figure 2: 9x9x9 Board, with red and blue Special Point spaces, and three completed plays.

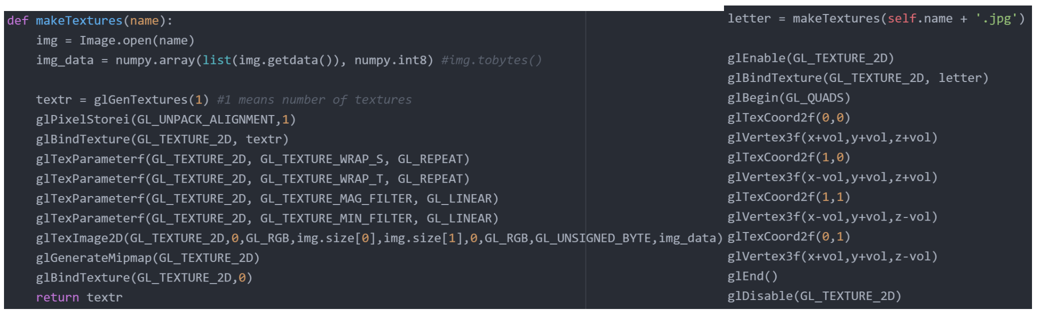
4.2 Obstacles

Unexpected obstacles threw my timeline far off course. I misjudged the requirements for a proof of concept, so while I planned to have my POC ready for the Seminar, it was not actually complete until the final week of the course. Extrapolation from a 2x2x2 lattice to an 9x9x9 lattice proved to be the simple step. I expected to finish the graphics about halfway through and then focus on a dictionary and improved user experience, but writing classes and finding appropriate tiles became a time sink of my project. My issues with classes stemmed from my lack of experience with Python, and I relied on documentation which, while helpful conceptually, is less useful than working examples. I spent all of Thanksgiving break attempting to achieve extruded letters, but to no avail. When Molly and Sasha recommended textures instead, it took a full week for us to find examples and debug them for my script. Upon reflection, I assumed I could achieve much more than I had the ability to, and did not account for necessary debugging time.



```
class Box(object):

    x = 0
    y = 0
    z = 0

    def __init__(self, centx, centy, centz):
        self.x = centx
        self.y = centy
        self.z = centz

    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def getZ(self):
        return self.z

    def draw(self):⋯

    def SpecPt(self, r, g, b, a):⋯

class Tile(object):⋯

class Player(object):⋯
```

Figure 3: My working Box class, with a Tile and a Player class below based off the successful Box.



```
def makeTextures(name):
    img = Image.open(name)
    img_data = numpy.array(list(img.getdata()), numpy.int8) #img.tobytes()

    textr = glGenTextures(1) #1 means number of textures
    glPixelStorei(GL_UNPACK_ALIGNMENT,1)
    glBindTexture(GL_TEXTURE_2D, textr)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,img.size[0],img.size[1],0,GL_RGB,GL_UNSIGNED_BYTE,img_data)
    glGenerateMipmap(GL_TEXTURE_2D)
    glBindTexture(GL_TEXTURE_2D,0)
    return textr
```

```
letter = makeTextures(self.name + '.jpg')

glEnable(GL_TEXTURE_2D)
glBindTexture(GL_TEXTURE_2D, letter)
glBegin(GL_QUADS)
glTexCoord2f(0,0)
glVertex3f(x+vol,y+vol,z+vol)
glTexCoord2f(1,0)
glVertex3f(x-vol,y+vol,z+vol)
glTexCoord2f(1,1)
glVertex3f(x-vol,y+vol,z-vol)
glTexCoord2f(0,1)
glVertex3f(x+vol,y+vol,z-vol)
glEnd()
glDisable(GL_TEXTURE_2D)
```

Figure 4: Properly-working texture mapping. The makeTextures() function, left, and a Tile's mapping to one side, right.

4.3 Improvements

If I were to keep working on this project, I would first figure out a way to address the small bugs that plague my final product. These are errors (tiles overwriting, selector not confined to the board, End Turn does not always reset completely) that I think I could fix given a small amount of extra time. Given a larger amount of extra time, I would write a hash function dictionary. Not only is this a challenging task, but detecting in a three-dimensional game which directions are allowed for words to spell in the proper order would be difficult. I was not able to figure out how to sense if words are connected or linear, so that would be the first challenge of the dictionary. After that, the function would have to check the letter arrangement both forward and backward, then put it through the hash function to see if the request matches with an acceptable word. After determining whether all words (including any hooks and two-letter words made) are legal, the dictionary would have to send illegal plays back to the rack. My project has the framework for the graphics of this, so I would expect writing the dictionary to be a code-intense module for my project. Some other abilities I would enjoy seeing would be blank tiles and tile exchanging. Finally, if this were a commercial product, I would do analysis and testing to see if my 9x9x9 board size is large enough, and if keeping the 100-letter tile distribution is appropriate.

References

[1] Wikipedia contributors. "Scrabble." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 28 Nov. 2016. Web. 28 Nov. 2016.

[2] Butts, Alfred Mosher. *Scrabble*. Pawtucket, RI: Hasbro, 2016. Board game instructions.