**Creating a Simple Maze Game Using Python Programming and the Uses of Python Programming for Game Development**

GISC 3200K

Summer 2014

Sarah E. S. Johnson

**Creating a Simple Maze Game Using Python Programming and the Uses of Python Programming for Game Development**

Python language has been used in many game applications and its functionality in regards to game development can be very beneficial both used alone and in conjunction with other programming languages such as C++. One well known game making company, Humongous Entertainment, decided to start using Python along with C++ for game programming over a decade ago. Bruce Dawson, Director of Technology for Humongous Entertainment, published a paper in 2002 describing some of the reasoning behind the switch, citing that “C++ is static, scripting languages are dynamic” meaning that “C++ runs fast, but scripting languages let you code faster.” Dawson also compared Python to Lua, another programming language Humongous was considering using for development. Humongous chose Python over Lua because of the much greater number of modules found in Python, the immense literature found on the programming language, and also because the game developers knew and liked the syntax.

My interest in creating a game using Python stems from my love of video games. Many of the older Nintendo games I love and enjoy, such as the original Legend of Zelda, are tile based games which can be navigated using Python code. Al Sweigart, a programmer who hosts a blog called Invent with Python, outlines how to navigate the over world of the Legend of Zelda using Python and Pygame in one post of his blog. Sweigart explains some of the tactics used when developing the code for what he deemed a “Walking Tour.” Sweigart also has all of his code and information about the walkthrough available for viewing on Github. Python specifically has been used by the above mentioned game developers at Humongous Entertainment, who helped develop a game called “Maniac Mansion” from Lucas Films that I also used to play on the original Nintendo Entertainment System. This type of programming for game development sparks a deep interest for me in particular, in learning more about programming languages and their uses.

The very first step in creating my Mazerunner game was to first create the actual maze itself using a text document in Notepad. The exterior and interior walls of the maze were created using ‘X’, with a ‘P’ denoting where the player is located and an ‘E’ showing the end of the maze. Other than just dead ends denoted with ‘X’, there were also dead ends containing a monster ‘M’ symbol and a ‘W’ symbol, both with individual messages telling the player that their path is blocked and to go back the other way. After creating the maze it was necessary to have Python ‘read’ the text document. To do this a function was created called “read\_maze” that used the built in ‘open’ function to open the maze document and the built in ‘readlines’ method to read the maze. The maze had to then be formatted by replacing split lines (‘\n’) with blank spaces so that when transferring the maze to python, the maze is read in the correct format. The next step was creating a function called “maze\_player” that prompts the player name by using the built in function ‘raw\_input’ to asks the player for his or her name. These first two functions set the basis for creating the actual game play.

A new function was then created called “find\_symbol” to locate the start and end points in the maze and can be used to locate other symbols in the maze as well. This function uses a ‘for’ statement in conjunction with an ‘if’ and ‘else’ statement to find a specified symbol. A somewhat reverse usage of this function was also created called “get\_symbol” that returns a symbol at a given coordinate. This function uses splicing to locate the symbol using ‘x’ and ‘y’ and the maze. To prevent the user from going off of the maze an ‘if’ statement is used to return the wall symbol ‘X’ if the player attempts to locate symbols outside of the maze document. The next step in the maze is to create a function that will allow the player to make sense of what their surroundings are, as well as introduce some of the obstacles in the maze. The function is called “surroundings” and it takes two inputs (‘coor’ and ‘maze’) to allow the player to know what their surroundings are. The function draws on basic knowledge of the coordinate plane, though the ‘x’ and ‘y’ values are reversed. By using the “get\_symbol” function, depending on the direction (‘N’, ‘S’, ‘E’ or ‘W’), +1 or – 1 can be added or subtracted to ‘x’ or ‘y’ to ascertain the current location of the player. Another function was then created called “message” that takes the ‘surrounds’ of the previous function and returns a message regarding the surround of the player. For examples “To the north you see a wall, to the south you see an open corridor, etc.” A unique message is given depending on the players surroundings. This function plays off of the previously built functions to work properly.

The next series of functions deal with actually moving the player through the maze. Though there is not visual representation of where the player actually is in the maze, the player can ascertain his or her surroundings and proceed in a direction that is not blocked. The next function called “valid\_moves” dictates the moves the player can make that will avoid obstacles. It defines ‘passable\_terrain’ to either open corridors or the end of the maze. The following function