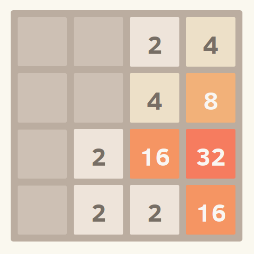


**3D 2048**

INTRODUCTION

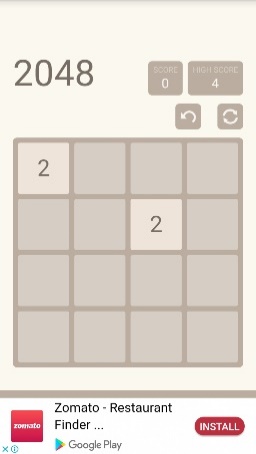
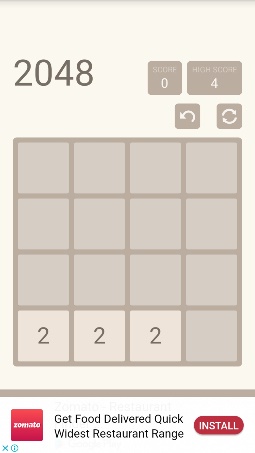
This game was made in order to utilise the Python programming concepts I had learned over the past two years and create a fun game which showcased the skills I had developed. The original version of this game involved long lines of cluttered, inefficient code which didn’t have any comments, but did the job. Having learned about classes, functions and file handling, I fixed these issues in the current version, added a functionality to save and load games, and introduced user selection of difficulty in the form of size of the grid as well.

**The Inspiration**

The original 2048 game consists of a 4x4 grid on which numbered tiles appear. The game begins with 2 tiles numbered 2. These tiles can be shifted up, down, left or right on each turn by the player. A tile numbered either 2 or 4 appears on a random empty spot after each turn. By merging like tiles, a tile with value double that of either tile is created. Thus, tiles on the grid always have value equal to some power of 2.

Rules

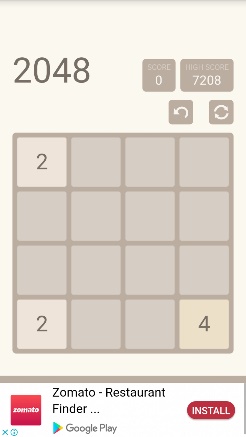
A tile moves until: a) It reaches the boundary

New tile

“DOWN”

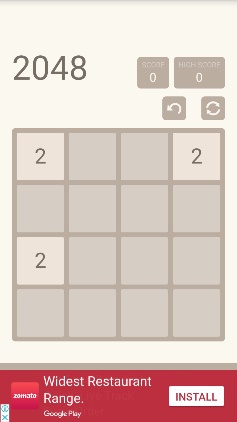
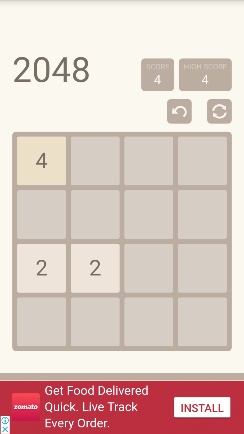
b) It hits another tile, in which case i) If it’s a differently valued tile, the tile stops right before this tile

New tile

“RIGHT”

ii) If it’s a tile with the same value, the tile merges with this tile to form a tile with the sum of both tiles (double of either, since they have the same value)

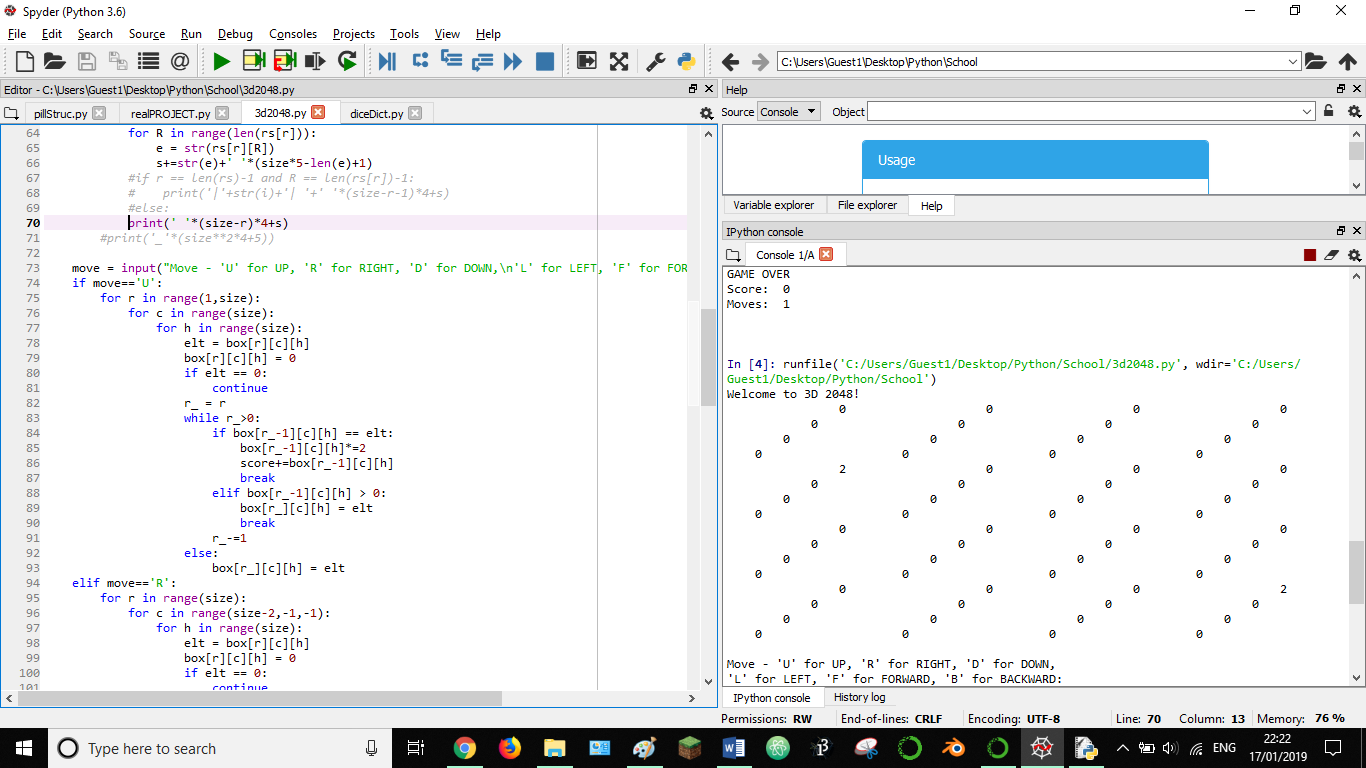
“LEFT”

New tile

A new tile is added every turn. The game ends if all the spaces are filled and no moves can be made. The goal of the game is to reach the 2048 tile (but the game continues after that).

**My Idea: A New Dimension**

In this project, I added a dimension of depth to the game, creating a 4x4x4 grid of tiles. The basic premise remained the same: numbered tiles appear on the grid, which could be moved around by the player, with the same rules as the original; the difference being that tiles could be shifted forward and backward.



< >

< >

‘U’, ‘D’

‘F’, ‘B’

‘L’, ‘R’

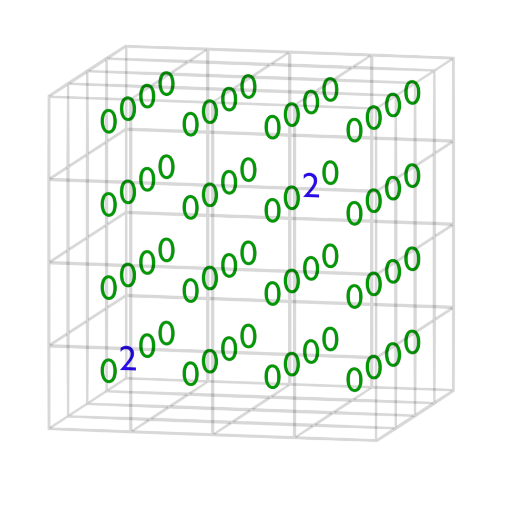
A column

A row

A ‘depth’

*Moves and Parts of the Grid*

The tiles are stored as follows: 4 tiles are stored in 1 list, 4 such lists are stored in 1 larger list, and 4 larger lists are in turn stored in 1 final list; thus a 3D grid of tiles is formed. Zeroes represent empty spaces. Using the random module, tiles are randomly placed on empty spots every turn.



*Visualising the Grid*

REQUIREMENTS

**Hardware**

* Keyboard (for controls and inputs)
* Monitor: 16:9, min. 1280x720
* Processor: 1GHz AMD Radeon or above

**Software**

* RAM: min. 20MB
* Hard disk space: min. 100KB
* OS: Windows 7 or above
* File containing python.exe should be added as path variable

MODULES USED

**Python modules:**

* Random: This module deals with introducing randomness in the program output. It

can be used to generate random integers or floats in a given range, to select a random element from a sequence, etc.

* Pickle: This module is used to handle binary files through Python. Objects can be

stored into such a file by conversion into bytes (serialization), and stored data can be retrieved by converting bytes back into objects (deserialization).

* Datetime: This module helps in obtaining and utilizing the current date and time of

the system. This is useful in adding timestamps, running a procedure at a particular date or time, etc.

**Custom module: classes.py**

This holds the templates for a ‘Box’ and a ‘Move’ class. The Box class defines the 3D grid which holds all the tiles, as well as functions to help in using and managing this grid, such as overloading the getitem and setitem operators to make accessing and changing a particular tile easier. The Move class defines what changes will be made to the grid for a specific move, and how those changes will be applied.

NON-PYTHON FILES IN THE 3D2048 FOLDER

* **saved.dat**

This is a binary file responsible for storing save states of games. Save states are elements of a single list object, which is serialized and stored as byte stream in this file.

Each save state consists of-

1. An instance of the Box module, which contains the tiles and size of the grid, the score and the number of moves as they were upon saving
2. The timestamp of the save

These save states can be loaded later on. While asking the user to select a save state to load, the size, score, number of moves, and timestamp are displayed to help identify the specific game that the user wants to play.

Thus, past games can be continued at a later date.

* **PLAY.bat**

This is a script file which runs a cmd prompt command and launches the game in terminal. So, user doesn’t have to run cmd and execute commands manually.

Script file source code

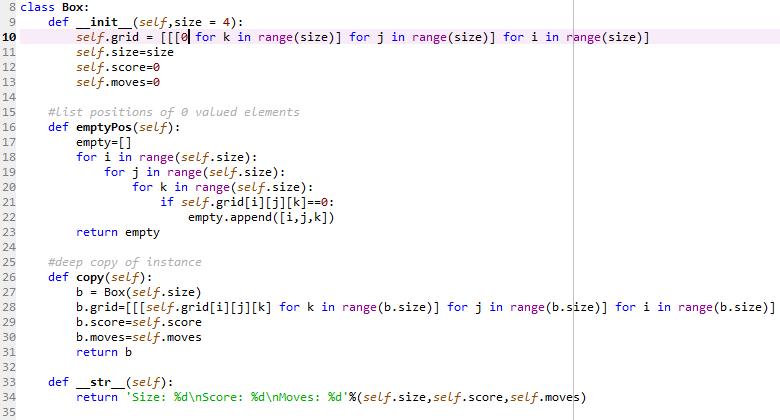
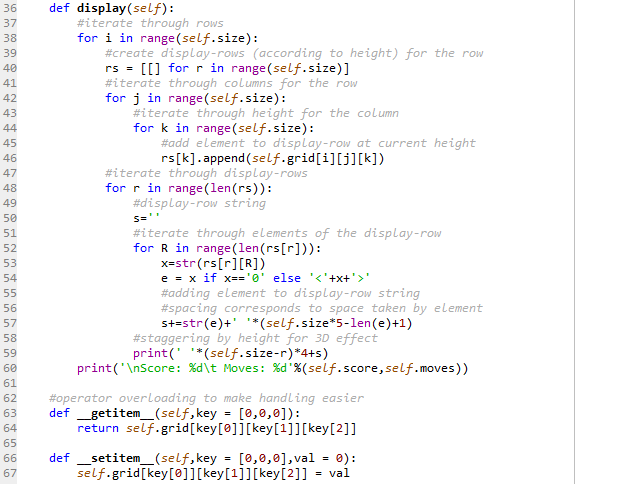


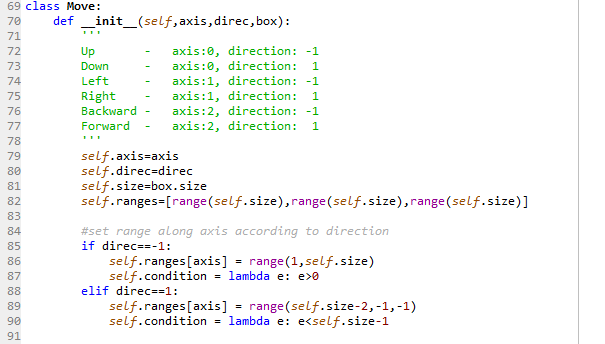
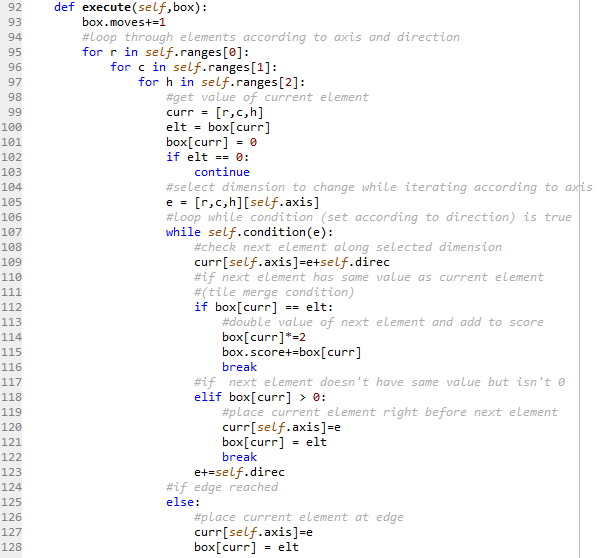
* **pycache**

It allows faster access from main.py to the classes.py module.

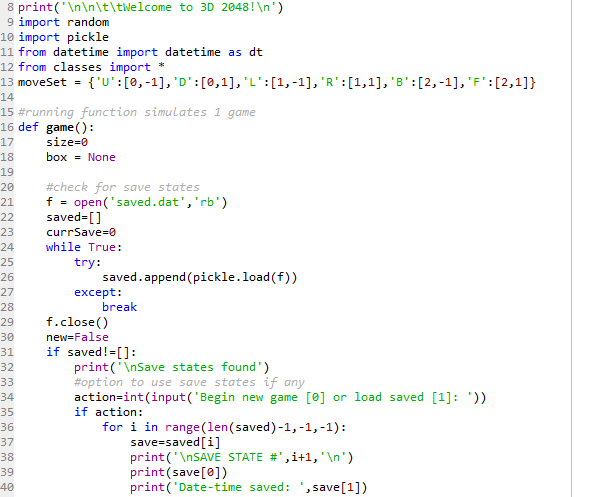
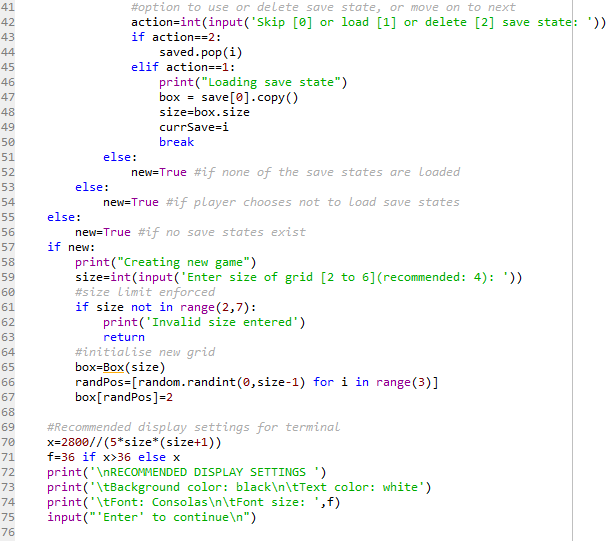
SOURCE CODE

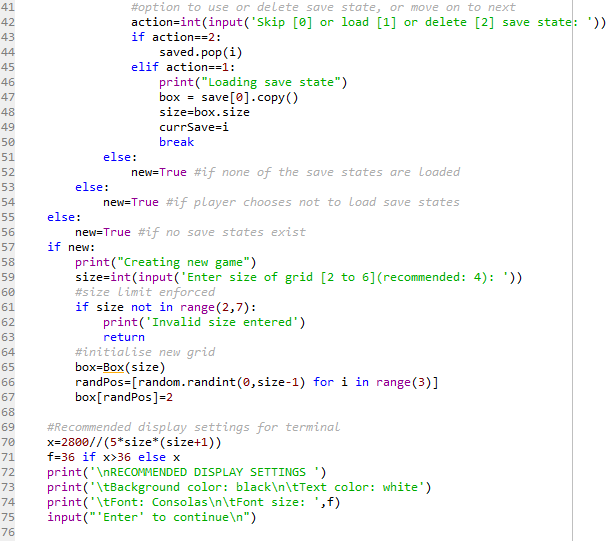
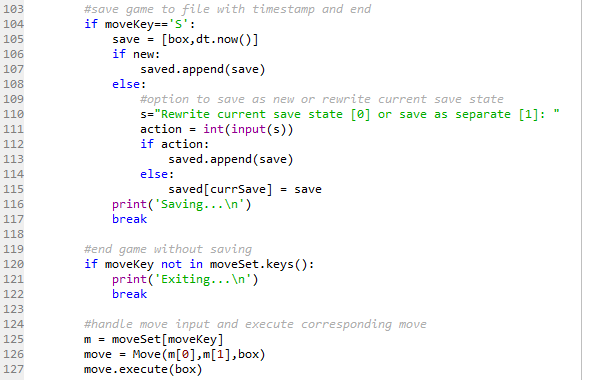
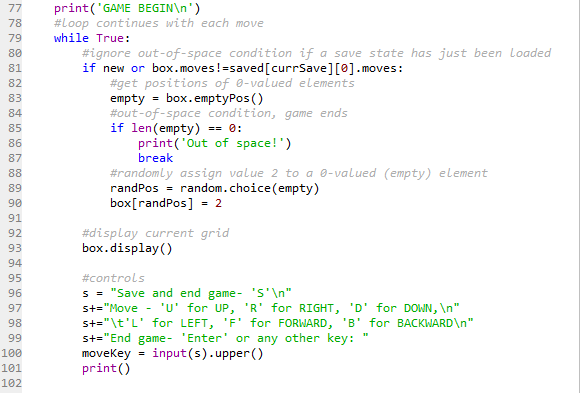
classes.py module

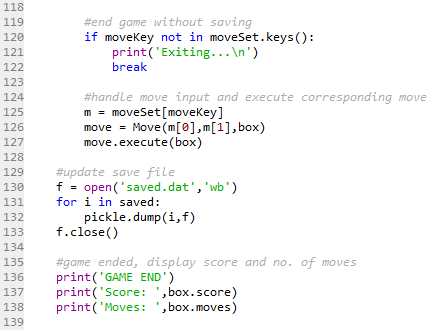
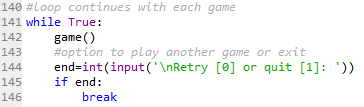
 

main.py

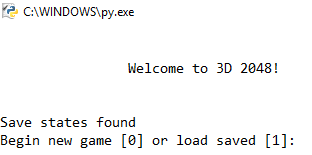
 

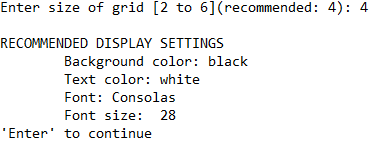
 

OUTPUT

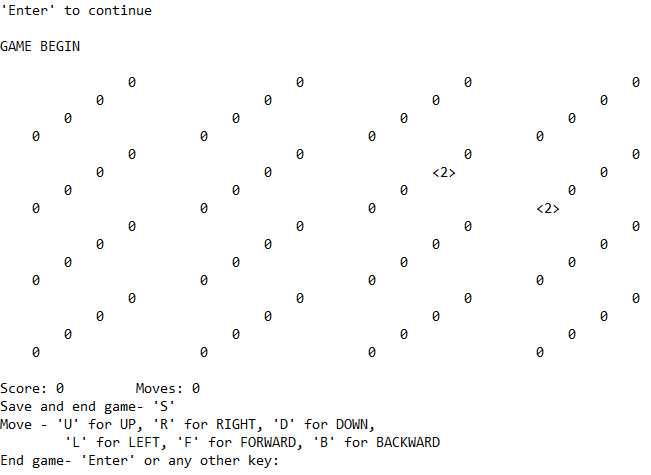
**Welcome Screen**



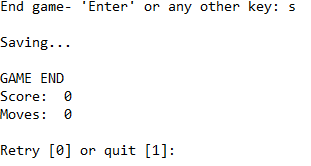
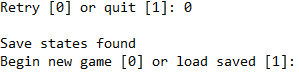
**New Game – Size (difficulty) selection Size Entered – Recommended Display Settings**

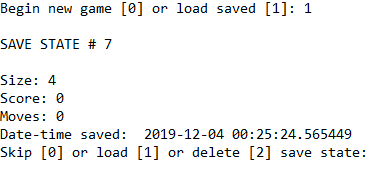
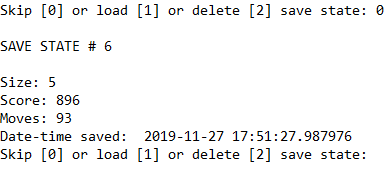
**Game Begin**



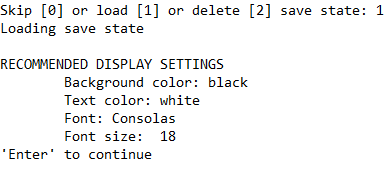
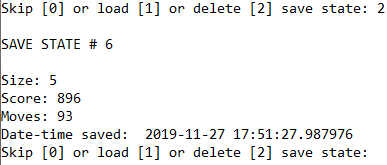
**Save Game – Game end, Retry option Retry Option**

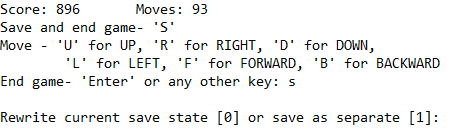
**Load Game – Save state handling Skip**

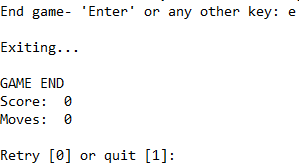
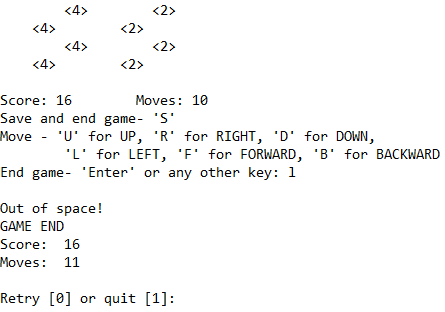
**Delete Load**



**Saving Loaded Save State**



**Exit Without Save – Game end, retry option Out of Space – Game end, retry option**

USER MANUAL

The game’s folder (which includes a classes.py, a main.py, a saved.dat, a PLAY.bat and a pycache) is saved in a user selected location. python.exe must be accesible through path variable. To play, one simply clicks on the ‘PLAY.bat’ file, which launches the game in terminal. User can then set appropriate display properties of terminal.

**Controls**

* ‘R’ corresponds to ‘right’ move
* ’L’ corresponds to ‘left’ move
* ’U’ corresponds to ‘up’ move
* ’D’ corresponds to ‘down’ move
* ’F’ corresponds to ‘forward’ move
* ’B’ correspond to the ‘backward’ move
* ‘S’ saves and ends the game
* ‘Enter’ or any other key ends the game

CONCLUSION

This game was designed on Spyder and is meant to be played on terminal. It has been tested and debugged thoroughly, and has been made more efficient repeatedly.

**Future Expansion**

* More user-friendly interface
* Better error handling
* Better visualisation/display and adding an element of animation
* Addition of sound effects and background music
* Introduction of variations/game modes to make the playing experience more interesting

BIBLIOGRAPHY

* Computer Science with Python for Class 12 by Sumita Arora
* W3Schools (<https://www.w3schools.com/python/>)
* Python Documentation (<https://docs.python.org/3/>)
* Tutorialspoint (<https://www.tutorialspoint.com/python/index.htm>)
* Stack Exchange (<https://stackexchange.com/>)
* Quora (<https://www.quora.com/>)