User manual - Tetris

Creators:

Seth Webb - Most of the game function in Jtetris
Paul Vetter - Brain and some High Score file development
Sarah RosinBaum - Graphic Design and turtle graphics
Anna Woodruff - No Contribution

Files

JTetris
- present the GUI(graphical User Interface) for tetris in a window and do animation while also running the game

JBrainTetris
- Similar to JTetris except that it uses a brain to play the game w/ out a human player

Brain
- simple heuristic logic that knows how to play the tetris

BrainTester
- test our brain and implement machine learning

Piece
- originally created this class because I thought we would have to do the rotation in here however we figured out that jtetris can rotate the pieces on its own so there is no need for this file. There is a class in jtetris that is used to create objects, such as current piece, for reference, built on this core idea though.

8-BIT WONDER
- Font file

Chip Tone Aggie War Hymn
- Music mp3 file

Chip Tone Aggie War Hymn short
- Music mp3 file

Texas A&M logo
- Image PNG File

Video Game medley
- Music mp3 file

Highscore
- Text file that holds our top scores and the players names

How To Play.

left arrow or "a" key moves piece left
right arrow or "d" key moves piece right
up arrow or "w" key rotates piece
holding down the down arrow or "s" key will cause the piece to fall twice as fast
Press the Spacebar to store a piece once per turn.
Press p to toggle pause.

When inputting New Highscore Name, DO NOT USE CONSOLE, just type it normally, backspace works, press enter to submit.
Valid inputs are all 26 letters of the alphabet, the delete key and the enter key.

DO NOT put in names longer than four characters!

Play Long Enough and discover the hidden musical easter egg!

How the code works:

JTetris

The program first imports random, pygame, and turtle.
Next it initializes pygame and a clock to be used for updates.
Global variables are declared and initialized in values.
A text font object is created from the 8-bit wonder font file with three size variations.
Next the highscore file is opened and passed into four pairs of two, each pair being a name and the score the name is associated with. We then close the txt file.
Next the aggiewarHymn is loaded into the music player and its volume is set.
Next two dimensional lists are created, one for each tetronimoe, each having an outer lists of possible rotations, with the inside list being values 0 or 1 based on if a block exists in that space.
Next a list of these 2d lists are created named shapes, and a corresponding list of colors is also created. Names of shapes came from the user handbook of the og tetris for nes.

Then the mainMenu function is called

mainMenu first hits play on the music mixer, starting the song.
It then calls the function MenuDisplay, which is our intro and it uses turtle graphics to display a logo and welcoming.
Next it creates a window of dimensions defined in global variables and sets it as the pygame display.
It next passes the window into the function main, after which when that function ends, mainMenu quits the display, since mainMenu acts as bookends for the program.

Main is as the name describes the main function of the program.
It Takes in a window, named screen.
Then Initializes a blank dictionary lockedPositions and then creates a grid based off of the dictionary, storing it in the variable grid.
It also initializes variables to fit the start conditions of the game, specific to the game loop.
It creates two elements of the class Piece, each having a position x,y, a shape, a color, and a rotation. This is done via getShape.
The getShape method returns an element of the piece class with x 5, y 0, and a random shape from the list shapes.
The game loop is initiated. The following is inside the game loop.
The music manager is the first thing evaluated in the game loop, it checks to see if the song is playing, if not, it starts up the song again, the song it starts up depends on how many playthroughs it's already done.
We also recreate the grid variable based on the dictionary of locked positions.
Next, the falling of the pieces is managed by set fall times and the clock rawtime,
whenever the current clock time is greater than the set fall check time, the piece is moved down one position. The set fall time is halved whenever fastFall is true.
Whenever the piece is moved down it checks to see it is valid using the function valid.
Valid creates a matrix of all black spaces in the grid and checks the positions of the current piece, if any of the positions of the current piece are not in the list of accepted pieces it returns false, and undoes the last change.
If valid is false it'' set change piece to true and stored this turn false, meaning you can store again.
If the user exits out of the window it will exit out of the game loop. Event type quit.
It then checks to see if the player is holding the down key, if so it sets fastFall to True, and False when they let go.
It also checks if the left or right key is pressed, if so it'll change the x value correspondingly, so long if the piece is valid, using the valid function.
If p is pressed, it will call the function pause.

The function pause takes the current screen and inside a loop reders the word pause onto it, it also pauses the music. The loop will end two ways, one by the user exiting the window causing the function to return false, or by pressing p again which exits the loop, resumes the music and returns true.
If pause is returned true the game loop continues as normal, if False, the game loop ends.
If spacebar is pressed and no piece is currently stored, the variable storedPiece is set equal to current piece, current piece to next piece, and then sets next piece equal to getShape.
It then sets isStored to true so from now on it'll behave differently, and also sets stored this turn to true so that you can't switch store infinitely.
If there already was a piece stored and you have not stored this turn, switches the values of current piece and stored piece using an intermediate variable, holdPiece.

If the up key is pressed, the rotational value of the current piece is increment by one, so long as it is valid, if not it is returned back to what it was.
A new list variable position is set equal to the return of the function convertShape, when the current Piece is passed as an argument.
The function convert shape creates an empty list, and gets the correct 2d list for shape based on the shape passed and the current rotation value of currentshape, cycling through using a modulus. Then using, enumerated form, It evaluates the list passed through for any values 1, ignoring 0. If the value is 1, the list positions gets appended by the current shapes position in the x and y, each being added by the index at which they were found in the list shapes defined at the begging(The values found by searching for 1's)
The list positions is then offset to account for how they shapes were originally defined, and for the fact that pieces originally exist above the play field.
The list is then returned back the game loop.
We then take the list of positions, evaluate every x,y coordinate pair in that list, and append the color at those points to the 2d list grid, at the same x,y point. Moving references of an object's color to an actual list of those color values, it'll make more sense when we clear lines.
If changePiece is true this cycle of the game loop, locked position is re-evaluated to now include the final placement of current piece, or shapePositions, by the color of them in those positions. It does this with an intermediate value p that is equal to the x,y values of shapePostions, and the color of the current piece referred to directly.
The current piece is then set equal to the next piece and the next piece is set to getShape. Change piece is reset to equal false for the next cycle of the game loop.
Since this is a the end of a turn, and the piece is in its final position, this is where we want to call clearRows to check if we need to clear rows.
The function clear rows takes the 2d list grid and the dictionary lockedPositions as arguments, The variable count is set to 0, this will count how many rows are cleared this turn. Starting from the bottom, a loop evaluates each line of the grid and checks to see if there is a black value in the line, if no black value exists, count is incremented and remember is set to the value of the row, i.

It also doing a try except, tries to remove all elements set to be in that line from the dictionary so they are no longer referenced, we do not set the grid values to black yet, even if missed here it'll be updated by the next increment of the game loop so no worries, the grid updates off of the dictionary. We then use the line clear count to increment the score, using if statements to determine the number of points to add to the global variable score.
We then do a lambda sort, sort by increasing values referenced second per element (in this case the row number of each point) If y, the height of each cube stored in the dictionary being referenced through the loop, is lees than that of the stored row height (less than means above since it references from top to bottom) it decreases (via addition) their height by count. This is kind of complicated especially since bigger numbers means lower position on the screen but in laymans terms, what it does is it looks at each and every square that is on the board, if that squares height was above the line(s) cleared, it lowers it by the number of lines cleared.
Next we call the rendering, or draw, functions. These are :

drawWindow - Takes in the window and grid(2d list of color values) as arguments. Draws a maroon background and the uses loops to cycle through all values of grid and draws the color value scored in a dimensional box of set dimension cell size. It also calls a function draw grid that draws gridlines to separate the play space into an easier to manage format, making it easier on players.

drawStoredShape - Takes in the window, shape of the stored piece if exists, and the boolean value check which is equal to isStored. It draws the word stored onto the window, and if check equals true, it draws the shape of the piece stored in the variable shape, from storedPiece. It does this by taking one of the loops used in convert shapes, but rather than adding it to a list it just renders it there.

drawNextShape - Takes in the window and the value of the variable nextPiece. It draws the word "next" and then, similar to drawStoredShape, renders the shape of the value in nextPiece that was passed into the function.

drawScore - This function takes in the window and draws the word "score" It also references and draws the global variable score. Note that the x position of where it draws the variable score changes depending on the number of characters used to express the score, this is done using a simple arithmetic to determine the length of score and shifts right accordingly.

drawHighScore- takes the window as an argument. References the global variables, four sets of two, each a name and score taken from a file, and it draws them respectively in a format that has corresponding names next to their scores.

Back in the game loop, after the draw calls, we then update the pygame display.

At the end of the game loop we check to see if the player has lost by calling the function checklist.
Checklist takes in the dictionary lockedPositions as an argument and checks through to see if any of the keys have any value less than 1, if so it returns true and the game loop is broken, false it continues through the game loop. We check locked positions which are only updated at the end of turns, this keeps the game from instantly losing when a piece is spawned over 1, which is a y value less than 1.

Outside the game loop, the mixer music is stopped.
Next, we check to see if we need to update the high score by seeing if the global variable score is greater than the least of the four highscores, if so we call the function newHighScore, which runs a loop similar to the game loop that looks for user raw input, for each raw input it concatenates it into a global variable name, and draws name for each passing of this loop. Once submitted by pressing enter, the loop breaks. Now that we have a value for the global variable name, we see just how many highscores the global variable score was greater than and overwrite the txt file accordingly, placing the global variable name along with the corresponding global score. We then close the txt file again and end the main function, sending us to mainMenu, which closes the display as previously stated, ending that function and the program.

JBrainTetris

Similar to JTetris except that it uses a brain to play the game w/ out a human player.
Instead of just asking for keyboard input JBrainTetris asks for input from the brain. In Addition to keyboard input it sends the brain the necessary parameters being the current board, x and y position of the piece in motion and its shape which the brain uses to calc the next move and returns it.

JTetris just requests input from the keyboard

```
291
292             for event in pygame.event.get(): #Pygame makes this so so sweet
293                 if event.type == pygame.QUIT:
294                     run = False #this breaks out of the while Loop
295
```

JBrainTetris also calls the brain and gets its input

```
242
243         for event in list(pygame.event.get()) + Brain.run(grid, currentPiece.x, currentPiece.y, currentPiece.shape):
244             if event.type == pygame.QUIT:
245                 run = False #this breaks out of the while Loop
246
```

Brain

The run method gets the grid and the x,y, and shape of the piece in motion from JBrainTetris. The brain has a counter so that it doesn't calculate a move every time otherwise it crashes the window. It creates an instance of the piece class with the given parameters. Sorts the grid into a matrix of ints because the grid that is given by JBrainTetris uses tuples for establishing the color of the blocks on the board. Brain then calls best_rotation which finds the best location out of the four possible rotations. Best_rotation also utilizes if statements so that it doesn't calculate the best position if the rotation is the same such as the cube. For each rotation it calls best_posistion which finds all of the valid positions for that rotation by calling validPositions. ValidPositions uses the getColumnHeight and valid methods to find all of the lowest possible positions on the board. Once it has all of the valid positions it calls rate which rates each of the positions based on if they will create holes, clear lines, their avg height, and max height. Rate returns a score for each position which best_position sorts through for each valid position and establishes the best one. Best_position then returns the score and x position of the best location to best_rotation. Best_rotation then calculates the best rotation based on each of their scores and returns the rotation and x position back to run. Run then figures out what input it has to return to JBrainTetris in order to get to the desired x position and rotation. Run will then return the desired input being either up(rotate), left, right, or down if it meets all of the other requirements.

```
274
275         r, position = best_rotation(grid, p) #gets the desired x pos and rotation
276
277         if r != rotation:
278             e = Event(pygame.KEYDOWN, pygame.K_UP)
279         elif position < x:
280             e = Event(pygame.KEYDOWN, pygame.K_LEFT)
281         elif position > x:
282             e = Event(pygame.KEYDOWN, pygame.K_RIGHT)
283         else: #if the piece is already in the desired position and rotation then i
284             e = Event(pygame.KEYDOWN, pygame.K_DOWN)
285
286        # e = Event(pygame.KEYDOWN, pygame.K_DOWN)
287         return [e]
```

BrainTester

This file is incomplete because I ran out of time so it just has some basic commands that I would have used. The brain tester would have used the scoreReturn method I wrote in JBrainTetris and the machineLearner method in the brain to test different values in the rate method and would have optimized them by running JBrainTetris with different values and finding the best ones for the highest score.