Technical note

tetris

Ainsley Nantes, Timothy Art, Chi-Lan Tran, Alex Kwong, Alex Luu, Sara Guo and Charlotte Zhen wrote this case solely to provide material for class discussion. The authors do not intend to illustrate either effective or ineffective handling of a managerial situation. The authors may have disguised certain names and other identifying information to protect confidentiality.

**Tetris Instructions**

Tetris is played by stacking Tetrominos to complete lines.[[1]](#footnote-1) Players earn points by clearing lines, and the objective of the game is to clear as many lines as possible and earn the highest score. There are the 7 different forms of Tetrominos in the game. Each has a unique colour and name.

**Exhibit 1:** Tetromino pieces with names below



To run the game, first extract the zip file, then open Tetris.v2.xlsm, and press the start button.

**Playing the Game**

The player must manipulate Tetrominos as they fall down onto the game board—moving them left and right across the board, as well as rotating the pieces to stack Tetrominos and fill up entire rows of the game board. When a piece encounters a barrier (either another piece or the bottom of the game board) the piece will stop falling and lock in place, retaining its form. At this point, the player will receive a new piece that will proceed to fall down the game board. This process repeats until the player “tops out,” at which point the game is over.[[2]](#footnote-2)

**Gaining Points**

When a player arranges Tetrominos in a manner that fills a horizontal line on the game board, the line will clear, removing all blocks from that row and shifting the remaining blocks in play down. Each time one or more lines is cleared the player earns points. Points are awarded progressively based on how many lines are cleared at one time. As a result, there is significant strategy involved with finding ways to build up lines and clear multiple lines at one time.

**Exhibit 2:** Progressive point system

Single Line Clear 40 Points

Double Line Clear 100 Points

Triple Line Clear 300 Points

Tetris[[3]](#footnote-3) 1200 Points

**Game Levels**

As the player clears more lines, the game level increases. When a level’s up, the level of difficulty increases causing the pieces to fall at a faster rate. There are a total of 12 levels in the game. Every time the player clears 10 lines the level will increase. More points are awarded for clearing lines at higher levels.

**Options Menu**

The player uses the options menu to select preferences for game play, which increase or decrease the game’s difficulty. The first option the player has is to select their starting level, up to a starting level of 10. Next, the player has four options: Holds, Wall Kicks, Hints, and Ghost. Selecting these options causes the following things to happen:

* **Holds**—Allows the player to hold a piece and swap it when desired
* **Wall Kicks**—An alternate rotation style that will shift the piece if it is spun against a wall and there is no space otherwise
* **Hints**—Allows the player to receive a suggestion from the computer as to an optimal position for a piece[[4]](#footnote-4)
* **Ghost**—Places a ghost piece in the position where the piece will be hard dropped[[5]](#footnote-5)

In the upper right hand corner there is a ‘Controls’ button. By clicking this button the game controls are displayed. When ready, the player presses the ‘Start’ button to begin.

**Exhibit 3:** Options Menu



**Game Controls**

The game is played on the keyboard, with the following controls:

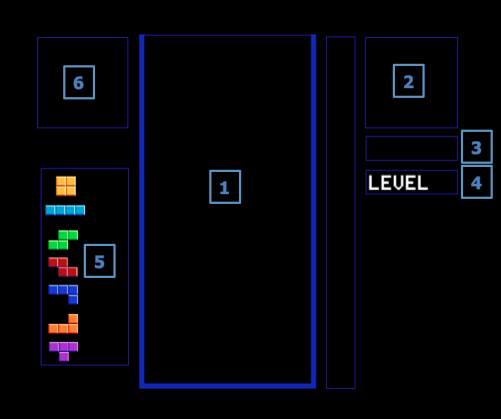
* **Left and Right Arrows**—Move the piece left or right across the game board
* **Down Arrow**—“Soft drop” the piece[[6]](#footnote-6)
* **Z Key**—Rotate the piece counter-clockwise
* **Up Arrow and X Key**—Rotate the piece clockwise
* **Shift**—Holds the active piece, if the ‘Hold’ option has been enabled. After holding a piece, the player must use the swapped piece
* **Space**—“Hard drops” the piece
* **S Key**—Pauses the game
* **H Key**—Asks the computer for a hint on where to place the next piece, if the ‘Hint’ option is enabled

**Exhibit 4:** Game Controls



**The Game Screen**

**Exhibit 5:** Tetris Game Screen



**Game Screen Labels:**

**[1] Game Board:** This is where the player will stack the pieces and clear lines

**[2] Next Piece:** This displays the next piece that the player will receive

**[3] Points:** This displays the current number of points the player has accumulated

**[4] Level:** This displays the Player’s current level. The bar on the side shows the player’s progress towards the next level.

[**5] Piece Counter:** This keeps track of how many of each type of Tetromino have been used

**[6] Held Piece:** This shows the player’s currently held piece (if any)

**Technical Note**

This note will take you through the logic and coding behind the game including:

1. Setting up the game board
2. Random piece generation
3. Piece movement and player input
4. The game timer
5. Hint calculation

**1. Setting up the game board**

Three classes were created to set up the game board: the grid class, the block class, and the Tetromino class.

**The Grid class**

The grid class creates an object representing one of the squares on the game grid. This is accomplished through using picture boxes on the userform, assigning them an integer value of 0 if empty, 1 if the grid unit contains a piece, and 2 if there are overlapping pieces in the grid unit. A 10 wide x 20 long 2D array was used to create these grids. This represents the game board and is displayed on the userform.

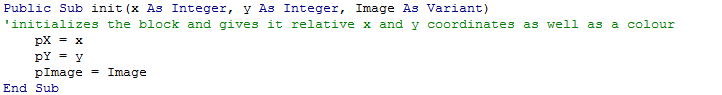
**Exhibit 6:** Creating individual blocks on the grid



**The Block Class**

This class represents a single block of a four-block Tetromino. It has a colour and a position relative to that of other blocks. The x and y coordinates are used to set its initial position. The image variable is used to pull the block of an assigned color from a saved picture file.

**Exhibit 7:** Positioning of Tetromino block components

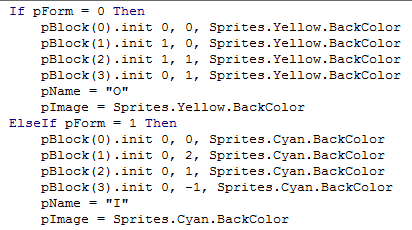


**The Tetromino Class**

The Tetromino class creates an object that represents a Tetromino on the game board. Each Tetromino piece is assigned to an integer that represents the form type (i.e. 0 = O, 1 = l, 2 = S, 3 = Z, 4 = L, 5 = J, and 6 = T). These form types are then built by using For Loops to assign positions to the top, bottom, left, and right coordinates.

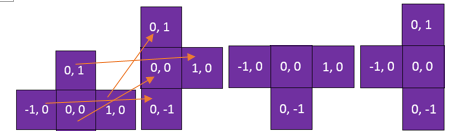
The Tetrominos are created in an initial position using an IF statement, drawn into spreadsheet cells on Excel, and loaded into an array. This prepares the pieces to be displayed on the game board when called upon.

**Exhibit 8:** Initializing the Tetrominos



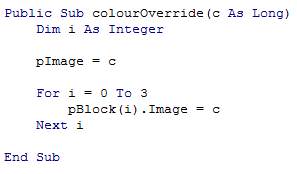
To spin the blocks from the initial position, an IF statement is used to alter the coordinates of the piece. The pattern for how the coordinates are changed is common across all Tetromino pieces, with the exception of the l and O shapes. Essentially what occurs is the x-coordinate becomes the negative of the original y-coordinate, and the y-coordinate value becomes that of the original x-coordinate. A similar, but reversed, formula was created to spin the Tetromino counter-clockwise.

**Exhibit 9:** Tetromino rotation logistics



Lastly, a sub was created to overide a piece’s default colur if given a new colour.

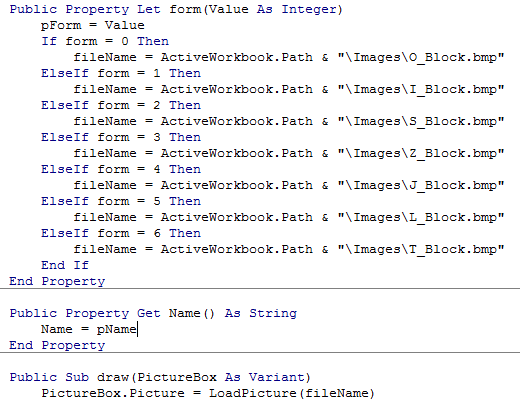
**Exhibit 10:** Assigning a new color to a Tetromino



**Tetromino Image Class**

This class creates an object that displays the images of Tetrominos onto the game board. The shapes created through the Tetromino Class are stored as images and loaded into ImageBoxes on the game userform.

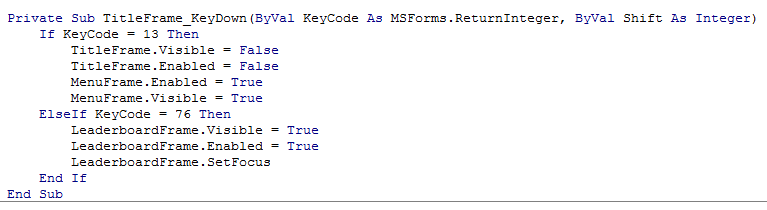
**Exhibit 11:** Pulling images of Tetrominos



**Userform**

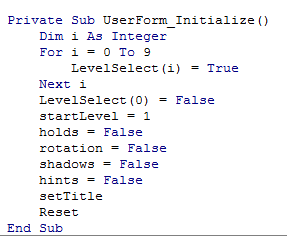
To display the game, a userform was used that allows the player to easily select their game preferences before starting to play. The userform consists of several frames including the title frame, menu frame, and leaderboard frame. When a specified action is performed, the corresponding frame becomes visible to the end user. For example, when the game is first launched, the user will press “enter” on their keyboard to continue to the next frame, where they will set their preferences.

**Exhibit 12:** Userform frame activation



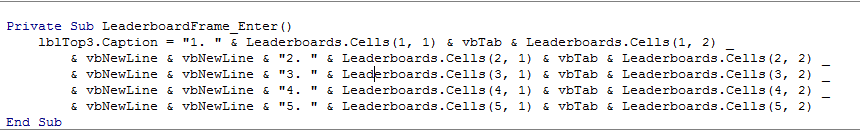
When the userform is first launched, all preferences are set to a default (i.e. player starts at level 1 and all additional options are turned off). Command buttons are then used to switch these options from their default selections.

**Exhibit 13:** Initializing the userform



The last frame that the player will access will be the Leaderboard screen. Players with top scores will be tracked on the “Leaderboards” worksheet and displayed in this frame.

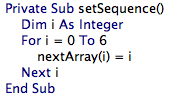
**Exhibit 14:** Building the leaderboard frame

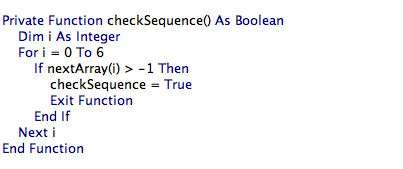


**2. Random Piece Generation**

Fair randomness in the generation of Tetrominos is important to the user experience. A simple RND() function would suffice for randomness but can be unfair because it has no constraints on generating the same piece consecutively. To combat the issue of uneven piece distribution, and to generate Tetrominos evenly, the pieces are drawn from a ‘bag’. Once a Tetromino is drawn from this bag, it cannot be drawn again from the same ‘bag’. The ‘bag’ is an array with 7 spaces and is named nextArray().There are corresponding numbers from 1 to 7 for each Tetromino.

**Exhibit 15:** setSequence() subroutine—refills the ‘bag’ with 7 Tetromino pieces

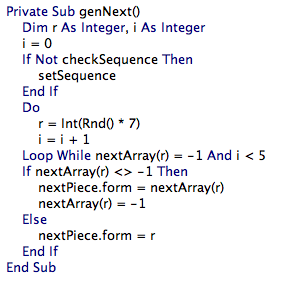
**Exhibit 16:** checkSequence() Function—checks if the ‘bag’ is empty



The genNext() sub will first check if the *bag* is empty, and refill the *bag* if it is. It then then randomly picks a number from 1 to 7 to choose a Tetromino from the *bag*.

The while loop is written so that it will choose a Tetromino up to four times. If it chooses a *taken* Tetromino all four times, the system will give up and generate a *taken* Tetromino for the player.[[7]](#footnote-7) As a result, the randomness of the piece generation is significantly better than if a simple RND() function had been used.

**Exhibit 17:** genNext() subroutine

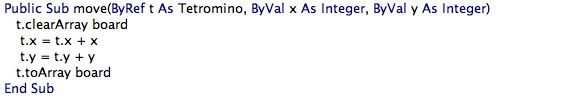


**3. Piece Movements and Player Input**

**Piece Movements and Bounds**

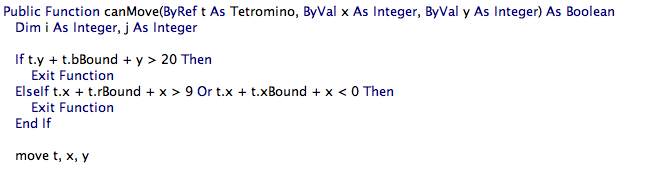
To move the Tetrominos, a sub was created that calls on the Tetromino class, “t”, the number of horizontal spaces inputted by the user, “x”, and the number of vertical spaces inputted, “y”. The sub then takes the Tetromino’s current x and y coordinates, and adds in the inputs to determine the Tetromino’s new position.

**Exhibit 18:** Move Function



To keep the movements of the Tetromino’s within the bounds of the game board an IF statement was used. The IF statement does not allow a move to be made if it will cause the new y position to be greater than 20, or the new x position to be greater than 9, or less than 0. This same canMove() function also checks for other pieces on the board, so that the Tetromino does not move if there is an obstacle in its way.

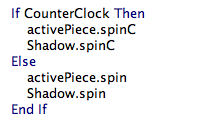
**Exhibit 19:** canMove Function



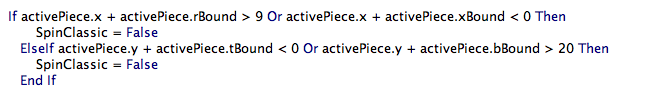
**Piece Rotation and Bounds**

The spin function spins a Tetromino either clockwise or counter-clockwise depending on the user’s input. Before setting a Tetromino in its new position, the function first tests whether the piece: a) goes out of bounds or b) hits another piece. If either of those 2 conditions are true, the piece will remain in its original position and will not spin.

**Exhibit 20:** Spin and Spin Counter-clockwise



**Exhibit 21:** Bound checking for rotation



**Wall Kicks**

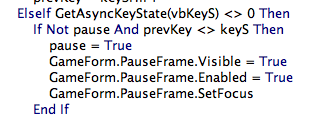
If the wall kicks option is on, the rotated Tetromino will be shifted either left or right, when it hits the right or left wall respectively.

**Player Input**

The subroutine checkKeys() is used to check player inputs and have the game act accordingly. The Windows API function for checking key presses was imported to allow key presses to be checked without interrupting the code. The Windows API function checks the inputs at a specific time, rather than always being on, which would have occurred if the onkey() function had been used instead.

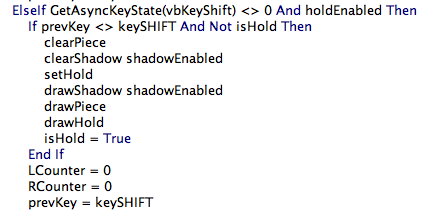
Based on the options a player selects prior to starting the game (i.e. starting level, wall kicks, holds, ghost, and hints) the program runs through a long series of IF statements that allow the game to function as intended. Whenever a player presses a key input that has a purpose in the game the actions associated with that key are enabled. For example, if the player hits “S”, which is the option for pause, the IF statement triggers the pause frame userform to appear, and the game is put on temporary hold.

**Exhibit 22:** Pause code portion of checkKeys() subroutine



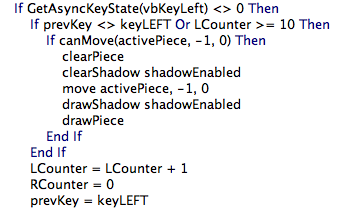
Another example is based on the player selections. If the player presses the “Shift” button the program checks if the player enabled the *Hold* option and if they did then the piece will be put in the hold box, and the previously held piece will be released. Similar IF statements exist for all keys involved in the game that the player could press.

**Exhibit 23:** Hold piece portion of checkkeys() subroutine

****

A unique portion of the code is how the program calculates at what point to hard-shift a Tetromino to the left or the right. [[8]](#footnote-8) It is based on how long the player holds down the left or right arrow keys. A counter variable is used to count up to 10, at which point the piece shifts to the farther possible left or right position. This is important because otherwise when a player clicked the right or left arrows the code would immediately shift the piece to the far side of the game board.

**Exhibit 24:** Shifting Tetromino to the left

****

**4. Game Timer**

The game timer is a timer class that was imported from the Windows API. It was imported because VBA does not offer a timer function that goes off in intervals of less than a second, which is not fast enough for Tetris. The timer is what handles the inputs from the player as well as the automated movement of the pieces down the game board. The timer is set to go off every 16 milliseconds; however, it realistically only ticks every 30 milliseconds because of the speed limitations of VBA. Upon each timer *tick*, the updateGame() subroutine is called.

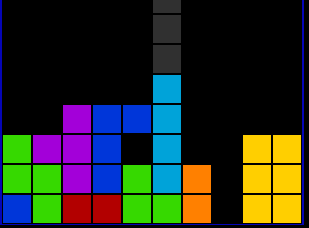
The updateGame() subroutine manages the game and inputs from the user. It checks to see if it is time to automatically drop the piece down a line through a counter variable. If the piece cannot be moved any further, it will set the piece in place and generate a new piece for the user. The timer also checks for user inputs. This is done through a second checkKeys() sub that looks at what key is being pressed and performs the related action. Lastly, the timer will check to see if the game is over. If a newly generated piece overlaps one of the already set pieces, this sets off a flag to the game timer letting it know that user has topped out and to bring the player to the ‘game over’ screen.

**5. Hint Calculation**

To calculate hints, the program tests all possible moves of the Tetromino—10 horizontal tiles with 4 rotations each—and then scores each of these moves. The scoring of the moves is based off of three factors, each with a different level of priority.

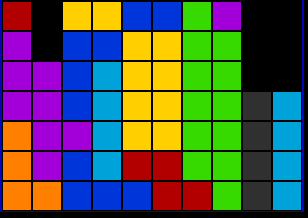
The first of these factors is whether or not the piece creates any new closed-off holes on the game board. This occurs when a piece blocks off an empty space on the board, forcing the player to clear all the lines above it before that line can be cleared. This factor is given the heaviest weight when scoring the move and the hint will avoid giving moves that create holes at all costs. To calculate the holes, the system first runs through the board and counts all the existing holes, then makes the move and recounts the number of holes. The more holes the move creates, the lower the score.

**Exhibit 25:** Hint calculator checking for holes



The next factor the algorithm looks at is how many lines the move can clear. This is fairly simple, where the score is linearly increased depending on how many lines are cleared. The scoring is also set up so that only when clearing 3 or more lines will its contribution surpass the holes in the board. To calculate this score the system makes the move and then counts how many lines are filled.

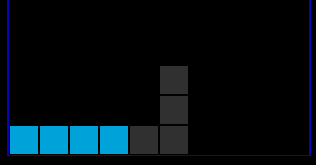
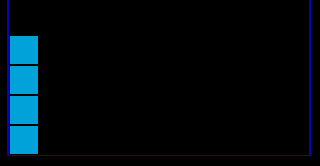
**Exhibit 26:** Hint calculator determining the number of lines that can be cleared



Clears 4 Lines

The last factor calculates how the piece fills the game board and prioritizes filling lower lines over higher lines. The logic behind this is because the optimal way to play Tetris is by filling the lower lines as much as possible and clearing them from the bottom to the top. To reflect this, the hint calculation makes the move and then sums up how the board is filled up with a multiplier for each line. This multiplier increases the further down on the game board the line is.

**Exhibit 27:** Hint calculator prioritizing filling lower lines over higher lines



>

The result of these three factors is a hint that can accurately pick out a good move for the player to make—simply because the *optimal* move is highly subjective to the player and their strategy.

**Conclusion**

Creating a program for Tetris in VBA creates an opportunity to build competence across many fundamental and advanced VBA tools. There are future opportunities to enhance the game through unique themes with special options, creating a two-player game, or a Tetris marathon, whereby the game continues well past level 12.

**Appendix 1: VBA Documentation for Reference**

**Game Module –** manages the game menus and inputs from the user

initializeGame(l As int, r As Boolean, h As Boolean, s As Boolean, hint As Boolean) – initializes the tetris game and settings based on the users input. L is the starting level, r is the rotation style (false = classic, true = with wall kicks), h is whether holds are enabled, s is whether piece shadows are enabled, and hint is whether hints are enabled.

terminateGame() – terminates the game timer and unloads the game form

checkKeys() – checks what key is being pressed by the user and performs the appropriate action

updateGame() – updates the game board. Checks if the right amount of time has passed before the piece moves down, checks for key presses, and checks if the game is over.

updateLeaderboard() – sorts the leaderboard in ascending order

**Tetris Module –** includes all of the functions needed for playing the game

setGameBoard() – sets up the worksheet as a grid, puts the tetris board on it, and clears the board array

setSequence() – sets the array of next pieces

checkSequence() – checks if the array of next pieces is used up

genNext() – generates the next piece in the game by randomly selecting one from the array, if the piece has already been taken, it selects another piece (retries up to four times before selecting a taken piece)

move(t as Tetromino, x, y as int) – will move the given Tetromino in a direction passed through the parameters.

spin(CounterClock as Boolean) – will rotate the Tetromino and bump it left or right if it hits out of bounds

spinClassic(CounterClock as boolean) – rotates the piece without any wall kicks

canMove(t as Tetromino, x, y as int) as Boolean – will check if the Tetromino can move in a direction or if it is blocked and return true or false

hardDrop(t as Tetromino) – will drop the given Tetromino as far as it will go and set it in place

updateCount() – updates the piece counter

setActive() – will set the active piece to the next piece, reset its position, and generate a new random piece

setHold() – will hold the active piece and either swap with the existing hold or go to the next piece

checkLines() as Boolean – will check if any lines are complete and return true or false

clearLines() – will clear all the completed lines and shift the lines down, should also increase the points with a multiplier based on how many lines were cleared at once

checkGameOver() as Boolean – checks if any pieces are hitting the top and will return true if that’s the case

setPiece() – will set a piece in place, check if any lines are clear, and check if the player has lost

shiftDown(index as int) – will shift the lines of the board down past a given row.

drawBoard() – draws the existing board

getHint() – gets a hint for the active piece

printBoard() – prints the board values onto the spreadsheet (for tracing and debugging)

drawPiece() – draws the active piece

clearPiece() – clears the active piece

drawNext(), clearNext(), drawHold(), clearHold(), drawShadow(), clearShadow(), drawHint(), clearHint()

**HintCalculation module –** Manages the functions for calculating and finding the hints

findHint(form as int, row as int, Hint as Tetromino) – sets the given Tetromino to the best hint location according to the algorithm

scorePosition(Hint as Tetromino, testBoard() as int) as double – returns a score based on the current position of the given Tetromino on the board

checkboard(a() as int, t as Tetromino) as Boolean – checks to see if the current position of the Tetromino overlaps another piece on the board (a).

checkHoles(a() as int, t as Tetromino) as Double – returns how many new holes the piece will create in the game board (a) if it is placed in its current position (a hole being a gap surrounded by placed blocks)

findTops(tops() as int, a() as int) – finds how many blocks have been placed in each column on the board (a) and fills an array (tops) with these values

countHoles(tops() as int, a() as int) as Double – returns how many holes are currently in a game board (a)

clearedLines(a() as int) as Double – returns how many lines are currently filled in a game board (a)

checkFill(a() as int) as Double – checks how filled the lines are and scores this with lower lines valued at more than higher lines.

**Tetrimono class** - an object representing a Tetromino on the game board

Properties:

Form – an integer corresponding to its shape (0 = O, 1 = I, 2 = S, 3 = Z, 4 = L, 5 = J, 6 = T)

Name – a string corresponding to its shape

x – the piece's x coordinate on the worksheet

y – the piece's y coordinate on the worksheet

i – The piece’s first index in an array

j – the piece’s second index in an array

bBound – the piece's bottom coordinate in relation to its registration point (0,0)

tBound – the piece’s top coordinate

rBound – the piece's rightmost coordinate in relation to its registration point

xBound – the piece's leftmost coordinate in relation to its registration point

Functions:

private setForm() – sets the piece's form according to its form property

init(Form as int) – initializes the piece given an integer for its form

draw() – will draw the piece at its x and y coordinate on the sheet

clear() – will clear the piece from the sheet

hintDraw() – draws the piece as an outline

hintClear() – clears the piece outline

toArray(a() as Grid) – will write the piece into a Grid array (adjusts the value

property)

toIntArray(a() as int) – will write the piece into a numerical array

clearArray(a() as Grid) – clears the piece from a Grid array

clearIntArray(a() as int) – clears the piece from a numerical array

spin – rotates the piece clockwise

spinC – rotates the piece counter clockwise

colourOverride(c as Long) – overrides the piece’s default colour given a new one

**TetromonoImage class** – an object that displays the images of the Tetrominos onto the game form

Properties:

Form – the integer corresponding to the piece’s shape

Name – the string corresponding to the piece’s shape

Functions:

Draw(PictureBox as variant) – draws the Tetromino into a picture box on the game form

Clear(PictureBox as variant) – clears the Tetromino from a picture box

**Block class** - an object that represents a single block in a Tetromino

Properties:

x – the block's x coordinate in relation to a registration block

y – the block's y coordinate in relation to a registration block

colour – the block's colour

Functions:

init(x as int, y as int, colour as long) – initializes the block at an x, y coordinate and sets its colour

draw(x as int, y as int) – draws the block given the location of a registration block

clear (x as int, y as int) – clears the block

hintDraw(x as int, y as int) – draws the block as an outline at the given location

hintClear(x as int, y as int) – clears the block outline

**Grid class –** an object representing one of the squares on the game grid, creates a picture box on the game form.

Properties:

Value – the integer value of the board. 1 = has piece, 0 = empty

Image – The colour of the piece filling the game board

Name – the name of the picture box on the userform

Functions:

setBorder (c as long) – sets the border colour of the grid square

clearBorder() – clears the border colour of the grid square

init(i as int, j as int) – initializes the game grid, setting the value to 0 and the image to the background colour of the board. The i and j is where the picture box will be drawn on the game from.

removeMe() – removes the picture box from the userform

**Timer** – a timer class that controls the game – updates 60 times per second.

Functions:

startTimer() – starts the timer

Class\_Terminate() – upon termination of the class, will stop the timer.

1. A Tetromino is made up of four blocks arranged next to each other. [↑](#footnote-ref-1)
2. To “top out” means that Tetrominos were stacked to the top of the game board and a new piece can no longer fall when it is generated. [↑](#footnote-ref-2)
3. A Tetris is when four lines are cleared in one move. [↑](#footnote-ref-3)
4. Using a hint causes the player to receive a point penalty because the level of difficulty is reduced [↑](#footnote-ref-4)
5. “Hard Drop” quickly moves the piece as far as it can go down vertically without hitting a barrier, or the bottom of the game board, and sets the piece in place [↑](#footnote-ref-5)
6. “Soft Drop” is used to increase the rate at which the piece moves down the game board. The piece moves one increment down every time the down arrow is pressed. [↑](#footnote-ref-6)
7. Note that a ‘taken’ piece is assigned a value of -1 [↑](#footnote-ref-7)
8. Hard shift to the left or the right is similar to a hard drop down in that it moves a Tetromino to the farthest possible left or right position on the game board before it hits another piece or the edge of the game board. [↑](#footnote-ref-8)