Pranjal Modi 5/22/23
AP Computer Science A Period 1

Final Project: Meeting #1 Document

Group Members' Name: Pranjal Modi
Group Title: Tetris Production
Project Title: Tetris in Processing
Brief Project Description: This project aims to recreate the game of Tetris in the Processing programming language; this implementation of the game aims to contain all of the defining features of the original version, with a built-in scoring system, rotation and movement support, and a wide variety of tetrominoes. At the start of the game, the user is presented with a completely empty board and a falling tetromino; as the tetromino falls, its rotation, *horizontal* position, and falling speed can be altered by the user. After it either hits the bottom of the board or another already fallen tetromino, another tetromino is generated, which undergoes the same process as the previous one. The game concludes when either time is called or the tetrominoes stack to the top; in order to maximize the score, the user is thus incentivized to completely fill out rows in the board, which will disappear afterward, shifting the remaining contents of the board down and leaving more space for future moves. The score will be determined by a weighted combination of the number of blocks successfully stacked and the number of rows successfully cleared, with the latter possessing more importance.
List of current functionalities:

- Full support for rotations, lateral movement, and randomized block spawning.
    - Each tetromino (excluding the I and O-shaped pieces, which are impractical for rotation) has four different possible *rotation states*, which can be altered by pressing the "A" and "D" keys.
    - Pressing the left or right arrow keys will move the tetromino in the corresponding direction.
    - Whenever a tetromino settles in its final position, a new tetromino of random rotation state and shape is spawned at a random horizontal position at the top of the board.
- Seamless board generation and updating.
    - Each space in the board is represented by a *Grid* object, with defined color and location; this helps simplify the coordinate system of the block and creates a more pixelated environment.
        - These objects can be updated when filled with a static tetromino.
- Full support for block falling, regeneration, and stabilization.
    - The *drop* procedure allows for blocks to fall at a multiple of a speed constant, and updates the grid accordingly.
    - A corresponding outline is generated below each tetromino that indicates where it will fall if its trajectory is left unaltered.

        - Once the tetromino reaches a location where it can stop movement, it sets the corresponding grid spaces to the *filled* status, locking their color in and making them part of the static region.
    - When a block is removed from the movement procedure, a new block is randomly generated.
- Support for user-directed speed changes via the up or down arrow keys.

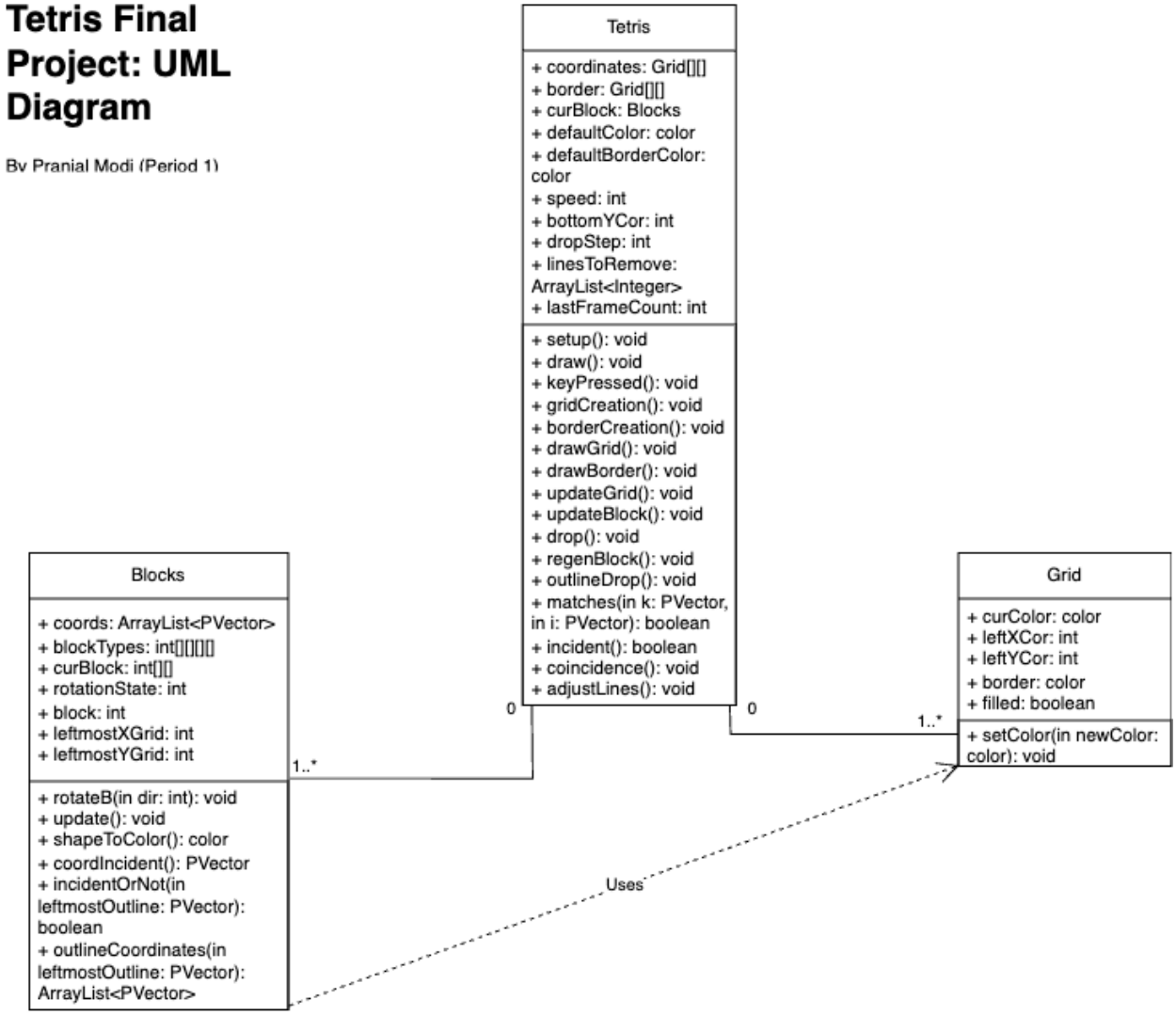
List of prospective functionalities:

- Developed scoring system and *visible display*.
    - Support for clearing out full rows and shifting down the rest of the board.
    - Support for combinations that further boost the overall score.
- Starting screen that regulates access to the game.
- Various different color schemes for tetrominoes which could be bought based on previous high scores (and that can provide scoring multipliers).
- Solving some glitches with the outline generation procedure for a falling tetromino.

Developmental Trouble: The bulk of my trouble in the development process so far has lied with the dropping portion of the game; I was able to code the various rotational states and blocks into the project relatively quickly and easily, but it proved extremely difficult to keep track of all the various conditions required to successfully process a full block drop. In order to solve this problem, I looked at it sequentially; I first began by implementing a procedure that would slowly move the *leftmost corner* of the current block down and then update the remaining coordinates of the block in kind, which was relatively simple to do. However, I ran into trouble when attempting to find a way of ending the drop of a block in response to its reaching a stopping condition (such as another dropped block or the bottom of the board). Using a conditional, I separated these two conditions from each other, allowing me to more easily tackle the first one through some looping constructs and alternate conditionals; for the second condition, I determined the first filled point beneath each box in the block, and considered the highest one of those points when determining the proper coordinates for the drop. After the drop was completed, I set the newly occupied grid spaces to have a filled value of *true* and then generated a new block, restarting the process. Although this overall procedure still contains a few bugs, it is orders of magnitude more efficient than the original version, making it more successful overall.

Note: The UML Diagram is on the *next page*.

UML Diagram:


Tetris Final Project: UML Diagram
By Pranial Modi (Period 1)
Tetris
+ coordinates: Grid[][]
+ border: Grid[][]
+ curBlock: Blocks
+ defaultColor: color
+ defaultBorderColor: color
+ speed: int
+ bottomYCor: int
+ dropStep: int
+ linesToRemove: ArrayList<Integer>
+ lastFrameCount: int
+ setup(): void
+ draw(): void
+ keyPressed(): void
+ gridCreation(): void
+ borderCreation(): void
+ drawGrid(): void
+ drawBorder(): void
+ updateGrid(): void
+ updateBlock(): void
+ drop(): void
+ regenBlock(): void
+ outlineDrop(): void
+ matches(in k: PVector, in i: PVector): boolean
+ incident(): boolean
+ coincidence(): void
+ adjustLines(): void
Blocks
+ coords: ArrayList<PVector>
+ blockTypes: int[][][][]
+ curBlock: int[][]
+ rotationState: int
+ block: int
+ leftmostXGrid: int
+ leftmostYGrid: int
+ rotateB(in dir: int): void
+ update(): void
+ shapeToColor(): color
+ coordIncident(): PVector
+ incidentOrNot(in leftmostOutline: PVector): boolean
+ outlineCoordinates(in leftmostOutline: PVector): ArrayList<PVector>
Grid
+ curColor: color
+ leftXCor: int
+ leftYCor: int
+ border: color
+ filled: boolean
+ setColor(in newColor: color): void
0
1..*
0
1..*
Uses