Pranjal Modi 5/30/23
AP Computer Science A Period 1

Final Project: Meeting #2 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. A *level-based advancement system* will also be included, with each level possessing a certain required score threshold; as the user moves through these levels, the game speed—and thus, difficulty—will increase exponentially, keeping the program interesting to play.
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.

Newly Added Functionalities:

- Full support for *line clears* and corresponding updates to the grid.
    - Upon the clearing of any number of lines, the affected lines will turn white for a short period of time, and then disappear; upon their disappearance, the blocks above each line will shift down, allowing for the board to remain cohesive.
- Full support for *scoring*.
    - Upon dropping a block, one point is given for each component of the block (i.e. four points for each successfully dropped block). If lines are also cleared out, the number of points given is equal to 5^n, where n represents the number of lines cleared at that exact moment (considering that $max(n) = 4$, the maximum score gained from line clearing at one time is thus $5^4 = 625$).
- User-oriented display system in the border of the game.
    - The score (maximum of six digits) is displayed in a blue box at the bottom of the screen. The middle-right of the screen also contains a graphic detailing the next three blocks to be dropped (with the top block being the next one).
- Active user input into game progress.
    - The red button in the top-left of the screen completely stops the game when pressed, while the gray button in the top-right of the screen pauses the game for some time when pressed.
- Almost fully glitchless runtime.
    - The vast majority of glitches in the game—mostly related to block placing and falling—have been patched, and users now have an almost fully glitch-free experience.
- Level-based progression system.
    - Upon the fulfillment of a particular score requirement, the current level increments by one; as the levels increase, the game becomes faster and the score requirements become higher. The score requirement grows logistically, while the speed increases exponentially.
        - This functionality still contains a number of small glitches, but the overall goals are still met.
- Support for an *instantaneous* block drop upon a mouse click on the game board.

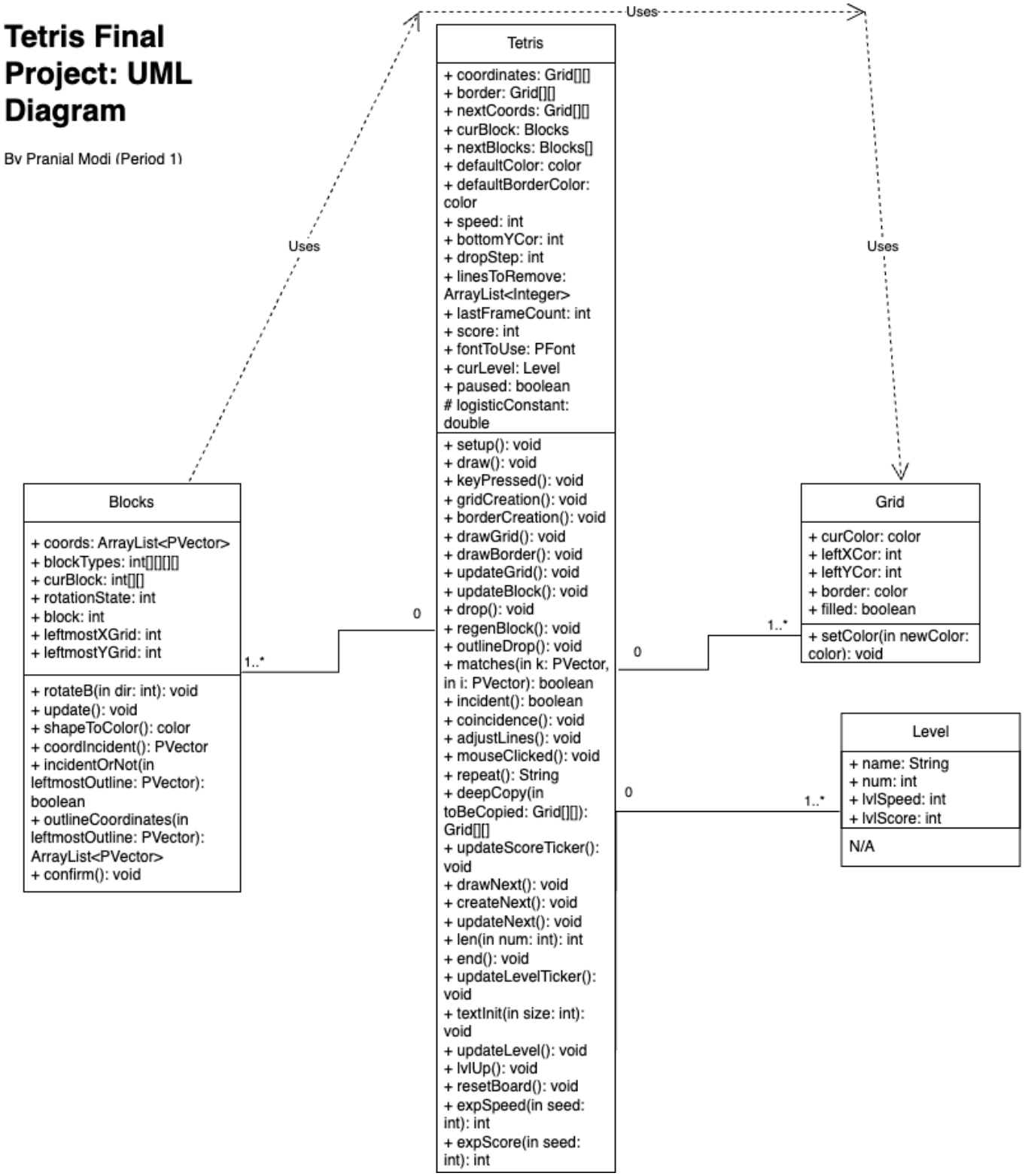
List of prospective functionalities:

- Game-over/Failure mechanics.
    - At this moment, the game simply throws an error when the tetrominoes go over the height limit of the screen; for the game to be more complete, however, a more integrated failure protocol should be implemented.
- [Not Completed Yet] Starting Screen for the game.
    - Although this functionality was listed in the previous document, I was unable to complete it to a satisfactory degree, as implementing this would require an extremely large-scale change to the initialization procedure of the game.
        - Possible Alternative: Level-based progression system.
- [Not Completed Yet] Different color schemes for tetrominoes.
    - This mechanic is associated with a starting screen for the game, so it still has not been implemented; I will further evaluate the feasibility of this once a starting screen has been properly implemented.
        - Possible Alternative: Changing tetromino colors with increasing levels.
- A more interactive and visually appealing border section.

Developmental Trouble: The majority of the developmental issues were faced during the first phase of the project (i.e. the production stage captured in the first document); these included difficulties in determining the right spot to stop a block's movement and in creating a reliable way to keep track of filled grid spaces. Although I was able to quasi-effectively solve these issues earlier, I ran into a significant number of glitches and special cases that threatened the overall viability of these functionalities during this period of development; fixing these glitches proved to be extremely troublesome, as they were often deep-rooted in the initial code. However, with the use of specifically targeted conditional statements and various confirmation and helper methods, the vast majority of these glitches were eventually patched (albeit only after significant testing and debugging). Another difficulty that I ran into involved the creation of a constantly updating text display system; since text in processing is of constant size, it often strayed out of its specifically allotted portion of the screen. In order to resolve this issue, I shifted down the text by a specifically determined value, centering it on the middle of the box; furthermore, I altered the size of the displayed text based on the number of characters in the associated string, allowing it to always fit within its box. Finally, I ran into trouble when trying to determine an accurate scaling model for speed and the scoring threshold as the current level increased; a linear model made the game too predictable and monotonous, while a quadratic or higher-degree function provided a very uneven sense of progression; to resolve this, I instituted a logistic growth model for the score (flattening out at the maximum six-digit integer value of 999,999) and an exponential *decay* model for the speed—flattening out at a value of four—as a lower speed value corresponds to a faster pace in the game. In all, I was largely able to successfully address the problems that I faced during development; although a few issues still invariably remain, I am very confident that I will be able to adequately address them in a later developmental stage.

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

UML Diagram:


Tetris Final Project: UML Diagram
By Pranial Modi (Period 1)
Uses
Uses
Uses
Tetris
+ coordinates: Grid[][]
+ border: Grid[][]
+ nextCoords: Grid[][]
+ curBlock: Blocks
+ nextBlocks: Blocks[]
+ defaultColor: color
+ defaultBorderColor: color
+ speed: int
+ bottomYCor: int
+ dropStep: int
+ linesToRemove: ArrayList<Integer>
+ lastFrameCount: int
+ score: int
+ fontToUse: PFont
+ curLevel: Level
+ paused: boolean
# logisticConstant: double
+ 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
+ mouseClicked(): void
+ repeat(): String
+ deepCopy(in toBeCopied: Grid[][]): Grid[][]
+ updateScoreTicker(): void
+ drawNext(): void
+ createNext(): void
+ updateNext(): void
+ len(in num: int): int
+ end(): void
+ updateLevelTicker(): void
+ textInit(in size: int): void
+ updateLevel(): void
+ lvlUp(): void
+ resetBoard(): void
+ expSpeed(in seed: int): int
+ expScore(in seed: int): int
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>
+ confirm(): void
Grid
+ curColor: color
+ leftXCor: int
+ leftYCor: int
+ border: color
+ filled: boolean
+ setColor(in newColor: color): void
Level
+ name: String
+ num: int
+ lvlSpeed: int
+ lvlScore: int
N/A
1..*
0
0
1..*
0
1..*