Pranjal Modi 6/5/23
AP Computer Science A Period 1

Final Project: Meeting #3 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. The game *will* be split into two different game modes: progression, where advancement is based on the number of levels completed (each with a unique speed and scoring requirement), and arcade (to be implemented), where the game continues until the blocks exceed the board's height limit. An animated starting screen displays the high score of the current session, along with the Tetris Logo; the play button also allows a user to start another game (later on, it will also allow them to choose between the two different game modes).

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.
- 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.

Newly Added Functionalities:

- Gamemode differentiation.
    - Although the arcade game mode has not been fully implemented yet, the *architecture* for supporting multiple game modes has already been introduced to the game.
        - Various *other* logistical game modes, such as the "Starting" and "Transition" game modes, have already been introduced alongside the active "Progression" mode.
    - Key procedures in the game's function—such as "draw" and "keyPressed", among others—have been simplified to make it easier to introduce different behaviors for different game modes.
- Fully operational starting screen.
    - Upon starting the game, the user is now presented with a starting/intermediate screen; this screen contains the session high score, a play button (currently only set to "Progression" mode), and an imported Tetris logo.
    - A gradient-based animation is also present in the middle of the screen, which continues as long as the screen is open.
    - In the game mode system, being on a starting screen is associated with the string "Starting".
- End of game handling.
    - Prior to this update, if a user stacked the blocks above the height limit of the board (a game-ending condition), the program simply produced an error, ruining the seamlessness of the project.
        - With this new functionality, however, losing the game will create a smooth transition to an intermediate "Game Over" screen, which will then lead back to the initial starting screen.
    - Clicking the red button at the upper-left-hand corner of the screen will also induce this transition to the starting screen.
- Smooth transitions between components of the game.
    - Whenever a user moves from a game to the starting screen (and vice versa), a grid-based transition is enabled, which slowly erases the previous screen and eventually replaces it with the new one.
- Full support for I-block rotation.
    - Although the rotation mechanics for the other blocks were stabilized near the outset of the project, the I-block, due to its unique length and shape, was not initially supported. However, it is now able to fully function in its horizontally-oriented state, and seamless rotation is possible.
- Internal Exception Handling System.

    - In an attempt to better process disruptions to the game, most of the directly-involved methods and components are declared to possibly throw exceptions; given the high frequency of nested methods in the code, these exceptions are eventually passed up to a select few methods, which then employ try-catch statements to address the issues. This centralization represents a drastic boost to the game's overall efficiency and performance.
- Full border interactivity.
    - The border section is now actively engaged with the user experience, allowing them to force an end to the game, temporarily pause it, or view the current level and score.

List of prospective functionalities:

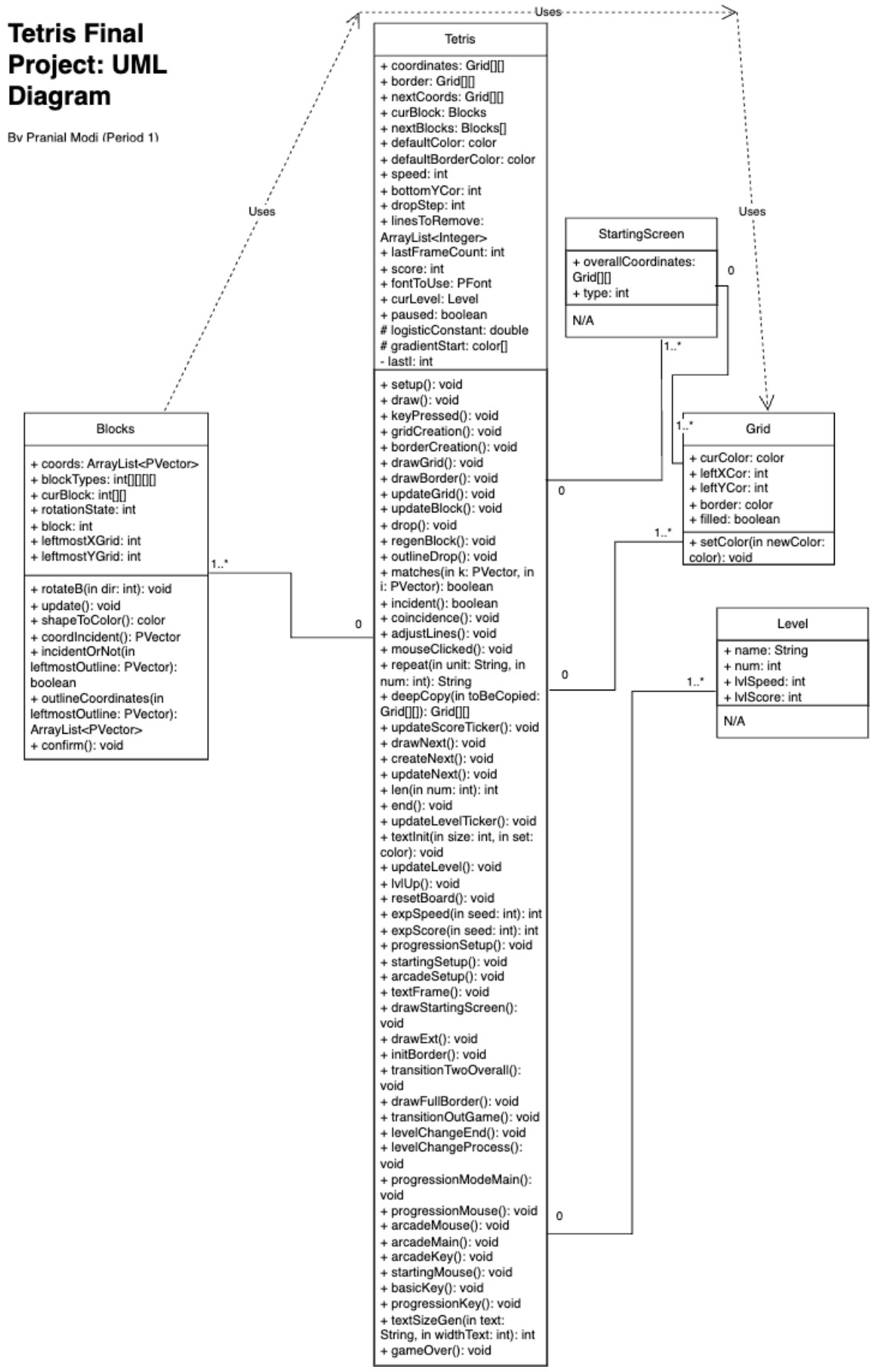
- Fully functional arcade game mode.
    - Although the *methods* for the arcade mode have been constructed, there is still some elaboration required. Furthermore, this game mode must be specifically integrated into the starting screen.
- Highest level tracker.
    - The starting screen could contain progress indicators for both the "Progression" and the "Arcade" game modes displayed next to each other, making it better representative of results.
- [Not Completed Yet] Changing color schemes for tetrominoes.
    - I am still determining the purpose of including this functionality; now that the starting screen is complete, I will try to see if this fits into the overall atmosphere of the game. *However,* I am leaning towards implementing a functionality that utilizes maximum level and/or maximum score to determine tetrominos' color.

Developmental Trouble: During this stage of production, the majority of the issues that I faced involved the creation and proper integration of the starting screen; before this update, running the program would automatically start a game of the "Progression" game mode, without any transition or intermediate interface. Changing this dynamic required a near-complete alteration of the setup and runtime components of the game, which proved extremely difficult due to the lack of abstraction in many key game methods (such as draw() and keyPressed()), where Progression-specific code cluttered the development environment and made the implementation of new game states very tedious; to remedy this, I encapsulated *all* "Progression" code into their respective methods (i.e. progressionMouse(), progressionModeMain(), etc.), clearing up space in some of the foundational methods to implement new game modes. By creating a defined game mode system, I was able to organize all the various states of the game into a cohesive and standardized system, where each mode—be it "Transition" or "Progression", among others—had its own defined behaviors and String-encoded identifier. Even after properly organizing the "Starting" and "Transition" states, however, I faced trouble properly generating the starting screen; the color gradient animation, specifically, was the most difficult, as I had to become proficient in using the lerpColor() method. Determining the exact location of the various

components of the starting screen (such as the Tetris logo, the play button, and the high score display) also proved to be quite tedious, requiring me to plan it out on a separate canvas application beforehand. However, I was able to eventually address all of these issues, and the current game is now easily able to accommodate transitions, intermediate and starting screens, and a multitude of different game modes.

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
# gradientStart: color[]
- lastI: 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
+ mouseClicked(): void
+ repeat(in unit: String, in num: int): 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, in set: color): void
+ updateLevel(): void
+ lvlUp(): void
+ resetBoard(): void
+ expSpeed(in seed: int): int
+ expScore(in seed: int): int
+ progressionSetup(): void
+ startingSetup(): void
+ arcadeSetup(): void
+ textFrame(): void
+ drawStartingScreen(): void
+ drawExt(): void
+ initBorder(): void
+ transitionTwoOverall(): void
+ drawFullBorder(): void
+ transitionOutGame(): void
+ levelChangeEnd(): void
+ levelChangeProcess(): void
+ progressionModeMain(): void
+ progressionMouse(): void
+ arcadeMouse(): void
+ arcadeMain(): void
+ arcadeKey(): void
+ startingMouse(): void
+ basicKey(): void
+ progressionKey(): void
+ textSizeGen(in text: String, in widthText: int): int
+ gameOver(): 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>
+ confirm(): void
StartingScreen
+ overallCoordinates: Grid[][]
+ type: int
N/A
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
1..*
0
1..*
0
1..*
0
1..*
0