Detailed Game Specification:  
TETRASLAM

Course: COMP 2659, Fall 2024  
Instructor: Paul Pospisil  
  
Author(s): Mack Bautista  
Last Modified: 12/09/2024

# General Game Overview

TETRASLAM is a redesigned version of the classic 1985 TETRIS, played on a 10x20 grid. The goal is to clear a pre-populated "failed state" field filled with a tower of tiles, which tracks both existing and newly placed tiles. The tower handles collision detection, row clearing, and tile counter updates. The grid is a 2D array where each cell represents a tile, either filled or empty. Active pieces (I, J, L, O, S, T, Z) are controlled by the player, who can move or drop them. When dropped, the piece is added to the tower, and the game checks for row clearance. Once a piece is placed, it resets to the middle top row instead of cycling back to the I-piece, allowing for strategic placement. The objective is to clear rows and reduce the tile counter to below 10/200. The game ends if the tower reaches the top of the grid. Naive gravity shifts cleared rows down, making space management crucial as non-cleared rows do not collapse. Players must avoid a scenario where no moves are available, which forces a restart. TETRASLAM challenges players with precision and strategy, focusing on efficient row clearing and tile counter management.

# The player’s active piece starts at the top-most row in the middle column. Since the I-piece is asymmetrical, it begins at the 5th column to allow symmetrical pieces (J, L, and O) to align properly with the grid. Active pieces cannot move beyond the 1st or 10th columns, ensuring they stay within the playable area of the grid.

# 2. Game Play Details for Core 1-Player Version

## Objectives and Rules

To win the game, players must reduce the tile counter—displayed on the right side of the screen—below 10/200 tiles in the grid. The game ends in a loss if a single tile reaches the top-most row of the playing field. Players must strategically ensure that there is enough space where new pieces are generated; otherwise, they will have no viable moves left, forcing them to restart the game.

The grid is generated close to the center of the 640x400 screen at (x, y) = (224, 32). The grid starts pre-populated with tiles beginning from the 5th row to accommodate the tallest piece, the I-piece. The first playable piece is always the I-piece, initialized at the top-left of the grid at position (x, y) = (288, 32). The primary objective is to clear as many tiles as possible from the tower while managing the tile counter.

The game begins with a 078/200 tile counter, meaning 78 tiles are already present in the grid. Each time a row is cleared, the tile counter decreases to reflect the updated total. For example, clearing one row reduces the counter to 068/200, and clearing two rows simultaneously reduces it further to 058/200.

The row-clearing mechanic uses naive gravity, adding an extra layer of difficulty. Instead of shifting all uncleared rows downwards until they collide with the highest remaining row, the naive gravity system simply shifts the tiles directly above the cleared row down by one row. This results in gaps or unevenly stacked tiles, requiring players to carefully plan their moves and account for these irregularities when positioning new pieces.

## Playable Piece and Playing Field Collisions

Playable pieces cannot move beyond the leftmost or rightmost boundaries of the playing field. When a block attempts to move past these stage boundaries, it is stopped, preventing further movement in that direction.

Each side of the active piece checks for collisions with both the playing field boundaries and the tower during movement. When moving left or right, the game uses the piece's current coordinates to detect collisions. However, the piece's position is only updated within the 10x20 grid indices after it is dropped, ensuring accurate placement within the tower.

## Playable Piece and Tower Tile Collisions

Collisions between a playable piece and the tower are essential for clearing rows. A row is cleared when all 10 columns in that row of the playing field are completely filled with tiles, leaving no empty spaces. The tower is made up of individual tiles, and for a row to be cleared, every position in the row must be occupied, either by tiles from the tower or by the newly dropped piece. Adjacent tiles must form a continuous line to trigger a row clearance. The game uses array traversal to check for full rows within the 10x20 grid, as the playing field is represented by a 2D array.

## Piece Movement and Selection

Players can use the arrow keys on the keyboard to move the active piece left or right across the 10x20 playing field. Holding an arrow key continuously sends a move request in the corresponding direction. Pressing the spacebar drops the currently selected piece into the playing field. To cycle through different pieces, players can press the "C" key. Holding this key cycles through the pieces in their predefined sequence.

If the player cycles to a piece that overlaps with the tower's tiles, the piece will automatically be adjusted or shifted to a valid position to prevent overlap. This ensures that the active piece can be cycled safely without immediately causing a collision with the existing tower structure.

## Objects

|  |  |  |  |
| --- | --- | --- | --- |
| Object or Object Type Name | Properties | Behaviours | Graphical Image |
| Tetromino  (I-piece [16x64], J-piece [32x46], L-piece [32x46], O-piece [32x32], S-piece [46x32], T-piece [46x32], Z-piece [46x32]) | * Starting position (x,y: 224, 30) * Size (width, height: I-piece [16x64], J-piece [32x46], L-piece [32x46], O-piece [32x32], S-piece [46x32], T-piece [46x32], Z-piece [46x32]) * Current index(integer: 0) * Horizontal and vertical velocity(integer constant: 16) * Array of player pieces (integer constant size: 7) * Tile count (integer, constant: 4) * Array layout(shape) of fixed size (4x4) * Merged (boolean flag) * Dropped (boolean flag) | * Move (left, right, or drop) * Update |  |
| Tower | * Tile count (integer: 79/200) * Grid 2-D array of fixed size (10x20) * Max row (integer: 0) * Is rows full (integer: 0) | * Clear * Update |  |
| Playing field  [160 x 320] | * Position (constant Integer pair: 224, 40) * Size (constant integer pair: 160, 320) |  |  |
| Tile counter | * Position (constant integer pair: fixed x,y - 416, 32) * Size (constant integer: 16) * Tile count (integer: (078 / 200) | * Update |  |

## 

## Asynchronous (Input) Events

|  |  |  |
| --- | --- | --- |
| Event Name | Triggering Input Event | Description |
| Move left request | Left arrow key is depressed | This will add the player’s playable piece’s horizontal velocity to -16. Velocity is constant as it moves per tile not by pixel. |
| Move right request | Right arrow key is depressed | This will add the player’s playable piece’s horizontal velocity to 16. Velocity is constant as it moves per tile not by pixel. |
| Drop block request | Space bar is depressed. | This will set the player’s active piece’s continuous vertical velocity to 16. It will stop after a collision is detected. |
| Cycle block request | “C” key is depressed. | This will set the initialized piece’s next element to the active piece. Inheriting the properties and incrementing the index counter. |
| Quit request | Esc key is depressed | The quit request is an asynchronous event that sets a user quit Boolean to TRUE. Since the game loop uses negative logic, this event signals the game loop to terminate gracefully by breaking out of the loop when the condition is met. |

## Synchronous (Timed) Events

|  |  |  |
| --- | --- | --- |
| Event Name | Trigger Timing | Description |
| Update music notes | Every 1/70th of a  second. | Time is retrieved from the CPU clock cycle via a function that invokes supervisor mode. This event updates the music notes based on the screen refresh rate. It ensures that the background music is synchronized with the visual updates of the game. Time tracking allows the game to precisely control the music notes ensuring they occur at regular intervals of 1/70th of a second based on the CPU’s clock cycle. |
| Render frame | Every 1/70th of a  second. | Time is retrieved from the CPU clock cycle via a function that invokes supervisor mode. This event renders the current game frame, updating the visuals on the screen to reflect the current state of the game. Time tracking allows the game to precisely control the frame rendering, ensuring they occur at regular intervals of 1/70th of a second based on the CPU’s clock cycle. |
| Processing requests | Every 1/70th of a second. | Time is retrieved from the CPU clock cycle via a function that invokes supervisor mode. This event processes any player input or game logic requests, ensuring that user inputs are handled in real-time with the screen refresh. Processing requests must occur before the frame is rendered otherwise the changes to the object’s properties will not be reflected. |

## Condition-Based (Cascaded) Events

|  |  |  |
| --- | --- | --- |
| Event Name | Triggering Condition | Description |
| Active piece - left edge playing field collision | The active piece moves left, and its position is greater than or equal to the left boundary of the playing field. | The active piece stops moving left when it is flush against the left boundary of the playing field. |
| Active piece - right edge playing field collision | The active piece moves right, and its position is less than or equal to the right boundary of the playing field. | The active piece stops moving right when it is flush against the right boundary of the playing field. |
| Active piece - bottom edge playing field collision | The active piece is dropped, and its layout array structure detects a collision with the bottom boundary of the playing field's grid.   |  | | --- | |  | | The active piece’s downward movement halts when it reaches the bottom of the playing field. |
| Active piece - top edge tower collision | The active piece's layout array structure overlaps with the top row of the tower in the playing field's grid. | The game detects that the active piece has collided with the top of the tower, potentially ending the game. |
|  |  |  |
| Tile counter increment | The active piece is dropped and added to the tower. | The tile counter increases by the number of tiles in the dropped piece, reflecting the current number of tiles in the playing field. |
| Tile counter decrement | A full row is cleared. | The tile counter decreases by the number of tiles cleared from the row, updating the count of tiles in the playing field. |
| Check rows if full | When the player drops an active piece. | When a piece is dropped, row clearance is checked starting from the piece's lowest row position (e.g., for an I-piece with a height of 4, this would be its bottom row). If an empty tile is found in the current row, the check moves upward to the next row. This process continues until a fully filled row is found or all rows occupied by the piece have been checked. |
| Clearing full rows | When the check rows condition checking is returned true. | When a fully filled row is detected, it is cleared by removing all tiles in that row. Tiles above the cleared row then shift downward to fill the gap, maintaining the integrity of the tower structure. The tile counter is updated accordingly, decreasing by the number of tiles cleared. This process may repeat if multiple rows become fully filled as tiles shift downward. |
| Shift cleared rows | The grid clears a full row, meaning all 10 columns in that row are filled with tiles. | When a row is cleared, the highest row of tiles in the grid (max\_row) is shifted downward by one position. This movement is applied to all rows above the cleared row. The array structure representing the grid is indexed starting from 0 at the top, with row indices increasing downward. The cleared row is removed from the grid, and the rows above it are moved down to fill the space. |
| Win or lose | The player wins if the tiles on the grid are less than 10 out of 200.  The player loses if the grid determines that at least one tile is at the top-most row. | Game winner flag is executed if the player succeeds in.  Game terminates if the player has at least one tile occupying the top-most row. |
|  |  |  |

## Hypothetical Gaming Session

|  |  |
| --- | --- |
| State of the game | Description |
|  | The game starts by loading a grid pre-filled with 78 tiles, representing the initial tower. The tile counter displays 078/200. The I-piece is generated as the first active piece, positioned at coordinates (x, y) = (288, 32).  The player moves the I-piece to the 2nd column and drops it, increasing the tile counter to 082/200. A new active piece is generated at the same starting position (288, 32). The player then moves this piece to the 9th column and drops it, bringing the tile counter to 086/200. |
|  | The third active I-piece is generated. To achieve the first row clearance, the player presses the "C" key three times to cycle through the pieces and selects the O-piece. The O-piece is dropped in the middle of the grid, reducing the tile counter to 068/200. |
|  | The player cycles back to the I-piece and drops them at the 3rd, 4th, 7th, and 8th column. The next row clearance occurs and the player is left with 086/200. With a 2-height gap, the player carefully cycles through available pieces with a height of 2 to fit into the left and right-most boundaries. Once out of the predicament, the player cycles back to the I-piece and drops it twice—once at the 1st column and once at the 10th column. Dropping the second I-piece results in another row clearance. The tile counter is now 084/200. |
|  | To clear another row, the player cycles to the O-piece and drops it in the middle of the playing field, positioning it between the 5th and 6th columns. This clears another row, and the tile counter decreases further. |
|  | The player’s patience is determined. Carefully dropping their chosen active pieces. The counter fluctuates— when a row clearance occurs the counter’s margin drops by a lot. Upon dropping a single piece though, the counter increases drastically. |
|  | As the game progresses, the player’s patience is tested. Carefully choosing and dropping pieces, the tile counter fluctuates. When a row is cleared, the counter decreases significantly, but each time a piece is dropped without clearing a row, the counter increases.  The player is now halfway through winning the game. Many repetitions and careful planning are needed as they continue to clear rows and manage their available pieces. |
|  | With only 014/200 tiles left in the grid, the player realizes that although it may seem like they are just a few moves away from winning, each piece drop to complete a row will cause the counter to increase. The game’s duration increases as the player cycles through their remaining pieces. |
|  | The player drops a final I-piece at the 7th column. The tile counter finally dips below 010/200, achieving the win condition. The screen clears, and the player’s excitement as their patience is rewarded with victory. |

# 3. Game Play Details for Core 2-Player Version

In the 2-player version, the rules remain the same as in the single-player game. Each player competes on their own 10x20 grid, with the objective of clearing tiles to reduce the tile counter below 10/200. The player who achieves this winning condition first is declared the winner. This introduces a competitive dynamic, as players must balance strategic tile placement, quick decision-making, and efficient row clearing to outpace their opponent.

|  |  |
| --- | --- |
| Player 1’s Screen | Player 2’s Screen |

# 4. Sound Effects

|  |  |  |
| --- | --- | --- |
| Sound Effect Name | Brief Description | Event which Triggers Playback |
| Dropping piece sound | thonk | player presses upon the action key to drop a block |
| Player bounds collision sound | tap | player collides with the left or right edge boundary field |
| Cleared row sound | bonk | player successfully clears all 10 columns |

The background music is “Theme A” from the original inspiration TETRIS.

# 5. Additional Features (Time Permitting)

1. **Winning Image:**

After the win condition is met (i.e., the tile counter drops below 010/200), display a celebratory winning image or animation.

* Display an image (like a trophy, fireworks, or a "You Win!" message) centered on the screen when the game is won.
* If possible, animate the screen with confetti or a series of visual effects, creating a more exciting conclusion to the game.

1. **More levels:**

* After the first level is cleared, load the next level with increased difficulty.

1. **Rotation system:**

* Each piece can now rotate. With a press of a key, the piece rotates 90 degrees clockwise.

1. **Retro mode:**

* Implement a retro mode that adjusts the game properties to mimic the original Tetris-inspired mechanics. This mode would require significantly more effort to change the object’s properties.