Amiel Iliesi

Mr. Winikka P2D1

9/14/16

Programming 2 / Applied Web Design

Tetris Senior Project (logic)

**(From main outline 3.)– what logic is required for Tetris to function?:**

1. Piece logic
   1. Rotation
      1. Disable rotation if the rotation direction is blocked by previously placed pieces/map boundary
         1. If rotation is block, try adjusting rotation set to the non-obstructed direction by 1 grid unit (attempt to adjust directional to allow rotation)
            1. If both directions are obstructed, then disable rotation entirely
      2. Follow input guidelines from b.ii
   2. Movement
      1. Disable moving left/right if boundary or placed block is obstructing movement in that direction
      2. Input by calculating output movement by pulsing input button then if held for longer than set time (0.5s?) set input repeater for every set cycle time (1/4?), but reset repeater status if input is released. Adjust held input priority according to 1.b.iii.\*
      3. Movement is coordinate set, modify direct coordinate values to skip vector addition to avoid dealing with roots and plane stretching.
   3. Placement
      1. Pressing (SPACE) or something similar to auto place to lowest possible grid unit.
   4. Hold/swap pieces
      1. Generate a list of next 5 pieces with an RNG
      2. Select next piece and swap places in list with current with swap button (X?)
      3. *Either* replace with current position, or allow 1 swap per piece emission and reset to top (to avoid spamming swap input to gain infinite time per move)
   5. Ensure easy detection of the piece and/or its blocks by global system for line completion detection or by other blocks to detect obstruction
2. Global logic
   1. Score
      1. See 5.b.
   2. Time (optional)
      1. Passive counting (for statistics or whatnot)
      2. Time count down mode (add win state with rounds that increase difficulty? Probably don’t add this)
   3. Piece auto-lower 1 grid timing
      1. Speeds up based off of:
         1. Placed pieces
         2. Lines completed (maybe?)
   4. When the game is failed
      1. Block is in state.placed above the map boundary
         1. To ensure that blocks have room to emit, then they must emit above, at the edge of the top boundary
            1. To ensure no out-of-bounds issue then raise the side boundaries up 4 units in the event that a user holds a directional to begin with and has a vertical line piece.
      2. OR have block in state.placed right below emission area at the top of map (pro, this is quicker; looks better, con, adjustment is needed for every possible next block spawn position in the (relative) center to detect fail for various states, e.g. if a sliver is available in the center and the next is a vertical line piece, but the game fails because a block was detected in the general area
   5. Game success?
      1. Round based, see 2.b.ii, optional/improbable addition
3. Boundary/map logic
   1. Interaction with 1.a-c.
   2. Contain the same variable that stops the block from entering its position by movement/rotation/placement BUT disclude the variable that registers the boundary as a block (in order to avoid random glitches/registers of map as an entity value
   3. Detection blocks for 2.d
4. UI/menu logic
   1. Menu
      1. Simple text/hitbox for cursor activation. Use this to activate global states
   2. Pause
      1. Pause timing/cycles and disable detection such as for game win/loss
      2. Use a screen obstructive sprite? Blur displacement map, or darken/tint
   3. Score/time display
      1. Text variable input. Add upper limit but disable further counting to avoid overflow
         1. Allocate enough space for max value OR scale text down if it surpasses allocated space
   4. Any HUD elements (that require logic to display)
      1. Hold pieces
      2. Score/time
      3. Next pieces
      4. Display cycling scalar (optional)
5. Any previous logic/inter-entity logic
   1. 1.d. and 4.d
   2. 2.a from 1.e and (main outline) 4.