Jacob Hartt
CS2300.001
Professor Sang-Yoon Chang
February 19 2023

Linear Domination Documentation

Program Description

The program will simulate a game using linear algebra concepts to draw lines onto a board. Each player will take turns making plays from an input file on the board. Illegal turns are ignored. The player with the most cells of their color on the board at the end of the game wins.

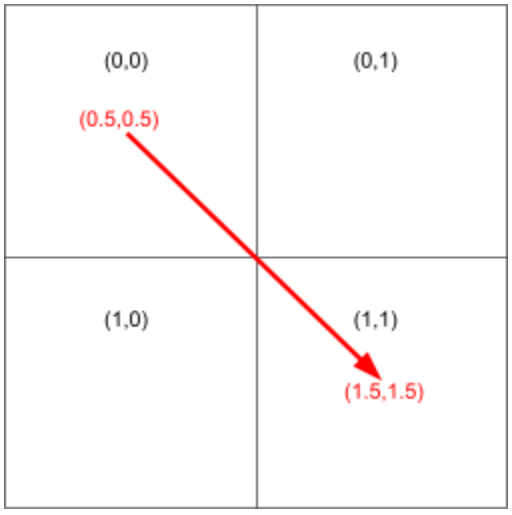
Design

Important Library Details

- Eigen
  - Library path: the headers for the Eigen library are located in /usr/include/eigen3 on my Linux machine.
  - Library version: I have installed Eigen version 3.4.0.

Approach

- Line Drawing: I found an algorithm called the Method of Amanatides and Woo outlined in a paper here as well as a more detailed document on github which appears perfect for the assignment. Essentially, lines are drawn by seeking out a line's intersections with cell borders and using those to move through a grid. This is one of the fastest perfect ray tracing algorithms, meaning that no microscopic overlaps with cells are missed. If an edge case needs to be implemented into the algorithm for when the line intersects a cell corner and not a cell wall to draw properly then I will implement it.
- The game will largely be handled by the RunGame() function which will be called in main.
- Coordinates: To the right is an illustration showing how the coordinates will work in this program: the coordinates in black are the cell coordinates, while the red coordinates are the absolute coordinates. As you can see, each cell (n,m) covers the coordinates $n \leq x \leq n + 1$ and $m \leq y \leq y + 1$ while the vector's end points cover the midpoints of each cell $(n + 0.5, m + 0.5)$.



Data Structures

- **TYPEDEF EIGEN VECTOR as alias POINT**
- **CLASS ParametricLine:**
    - Attributes:
        - EIGEN VECTOR vector_
        - EIGEN VECTOR vector_normalized_
        - POINT tail_cell_
        - POINT head_cell_
        - POINT tail_point_
        - POINT head_point_
        - POINT mid_point_
    - Constructor(tail_cell_x, tail_cell_y, head_cell_x, head_cell_y): The constructor will first set the tail_cell_, head_cell_, tail_point_ and head_point_, vector_, vector_normalized_ and mid_point_ (using BarycentricCoordinate(0.5)), variables in that order.
    - Non-getter/setter Behaviors:
        - FUNCTION BarycentricCooridinate(t): calculates a barycentric coordinate on the line.
- **CLASS Game:**
    - Attributes:
        - BOARD board_
        - STL STACK OF PARAMETRICLINES plays_
        - UNSIGNED INT prev_plays_checked_
        - STRING output_file_path_
    - Constructor(input_file_path, output_file_path): parses the input. Sets board_ to an NxN 2d stl array full of ‘ ‘ chars, sets plays to the list of plays turned into parametric lines, sets output_file_path_ to output_file_path and sets prev_plays_checked to K.
    - Non-getter/setter Behaviors:
        - FUNCTION Play(): plays the game. Contains the main logic for the game.
        - FUNCTION isPlayValid(play_num): checks if the cell coordinates of both endpoints of the play at play_num are within the board, if the midpoint of play is not the same as the midpoints of the last prev_plays_checked_ plays, and if play is not perpendicular to the last prev_plays_checked_ plays.
- **CLASS Board:**
    - Attributes:
        - 2D STL CHAR ARRAY grid_
    - Constructor(width): The method will initialize grid_ to be a width by width 2D STL array of chars filled with ‘ ‘.
    - Non-getter/setter Behaviors:

        - FUNCTION CountColor(color): counts the number of cells in grid_ colored as color
        - FUNCTION DisplayBoard(): prints out the contents of grid_
        - FUNCTION PlotLine(line, color): plots a line onto the grid using the Amanatides and Woo algorithm.
    - Getters/Setter Behaviors:
        - FUNCTION setCell(color, row, col): sets the element of the grid at (row, col) to be color.
        - FUNCTION getGrid(): retrieves and sends the entire grid_

Input/Output Format

- Inputs:
    - ```
      N K
      sr1 sc1 er1 ec1
      sr2 sc2 er2 ec2
      ...
      srL scL erL ecL
      ```
- Outputs:
    - ```
      <N by N board>
      Player X: V cells; Player O: W cells
      ```

Testing and Evaluation Strategy

As a general rule of thumb, each method will be tested before it is implemented into other, untested methods or to main.

1. ParametricLine will be tested first, as it is at the bottom of the class hierarchy.
    a. The constructor will be tested first by initializing everything BUT the midpoint and viewing a line's contents with the debugger.
    b. BarycentricCoordinate will then be tested by temporarily making the method public and calling it from main.
2. Board will then be tested second. The testing methods will involve both the debugger and terminal output.
    a. The debugger will be used to test the constructor, getCell(), and DisplayBoard() methods.
    b. The terminal will be used to test everything else.
    c. PlotLine() will need the most extensive and thorough testing of any part of the program due to its complexity. This should be implemented after the rest of Board.
3. The game class is then tested.
    a. The constructor will be tested first and the contents of created objects will be viewed in the debugger.
    b. IsPlayValid() will then be tested.
    c. Lastly, Play() will be tested.

4. Finally, main will be tested by creating a game object and running Play().

Other Design Choices

- Eigen vectors were typedefed as points to simplify calculations.
- If necessary, a tolerance will be added to the t_max_x == t_max_y condition to ensure that any errors in floating point calculations will be addressed.

Pseudocode

```
TYPEDEF Eigen::Vector as Point

DEFINE kCellMidpointX as 0.5
DEFINE kCellMidpointY as 0.5
DEFINE kBarycentricMidpoint as 0.5
DEFINE kEmptyCell as ‘ ‘
DEFINE kBlackCell as ‘X’
DEFINE kWhiteCell as ‘O‘

CLASS ParametricLine:
   PUBLIC:
      CONSTRUCTOR ParametricLine(tail_cell_x, tail_cell_y, head_cell_x,
                                 head_cell_y):
            SET head_cell_ to {head_cell_x, head_cell_y}
            SET tail_cell_ to {tail_cell_x, tail_cell_y}

            SET head_point_ to {head_cell_x + kCellMidpointX,
                                 head_cell_y + kCellMidpointY}
            SET tail_point_ to {tail_cell_x + kCellMidpointX,
                                 head_tail_y + kCellMidpointY}

            SET vector_ to head_point_ - tail_point_
            SET vector_normalized_ to vector_ / vector_.magnitude

            SET midpoint_ to BarycentricCoordinate(kBarycentricMidpoint)

      FUNCTION BarycentricCoordinate(t):
            RETURN ( (1 - t) * tail_point_ ) + (t * head_point_)
```

```
      // getters and setters

   PRIVATE:
      INSTANCE VARIABLE Eigen Vector vector_
      INSTANCE VARIABLE Eigen Vector vector_normalized_
      INSTANCE VARIABLE Point head_cell_
      INSTANCE VARIABLE Point tail_cell_
      INSTANCE VARIABLE Point head_point_
      INSTANCE VARIABLE Point tail_point_
      INSTANCE VARIABLE Point midpoint_


CLASS Board:
   PUBLIC:
      CONSTRUCTOR Board(width):
            ASSIGN a width by width 2d array to grid_

            ITERATE through grid_:
                  SET the current element to ‘ ‘

      FUNCTION CountColor(color):
            DECLARE an integer count as 0

            ITERATE through grid_:
                  IF the current element is equal to color:
                        INCREMENT count by 1

            RETURN count

      FUNCTION DisplayBoard():
            ITERATE through the rows of grid_:
                  ITERATE through each element of each row:
                        PRINT out the element and a space after it

                  PRINT a newline character

      FUNCTION PlotLine(line, color):
            SET the grid_ element at line.GetTailCell() to color

            IF line.GetVector does not equal {0,0}:
                  DECLARE x as line.GetTailCell()[0]
                  DECLARE y as line.GetTailCell()[1]
```

```
DECLARE x_step
IF line.GetVector()[0] > 0:
      SET x_step to 1
ELSE IF line.GetVector()[0] < 0:
      SET x_step to -1
ELSE:
      SET x_step to 0

DECLARE y_step
IF line.GetVector()[1] > 0:
      SET y_step to 1
ELSE IF line.GetVector()[1] < 0:
      SET y_step to -1
ELSE:
      SET y_step to 1

DECLARE t_delta_x as 1 / line.GetVectorNormalized()[0]
DECLARE t_delta_y as 1 / line.GetVectorNormalized()[1]

DECLARE t_max_x as kCellMidpointX /
                  line.GetVectorNormalized()[0]
DECLARE t_max_y as kCellMidpointY /
                  line.GetVectorNormalized()[1]

DO:
      IF t_max_x > t_max_y:
            ADD t_delta_x to t_max_x
            ADD x_step to x
            SET the grid_[x][y] to color
      ELSE IF t_max_x < t_max_y:
            ADD t_delta_y to t_max_y
            ADD y_step to y
            SET the grid_[x][y] to color
      // Safety condition which might not be necessary
      ELSE:
            ADD t_delta_x to t_max_x
            ADD t_delta_y to t_max_y
            ADD x_step to x
            ADD y_step to y
            SET the grid_[x][y] to color
```

```
                  WHILE {x,y} does not equal line.GetHeadCell()

      // Getters and setters

   PRIVATE:
      INSTANCE VARIABLE 2D Std::Array of Chars grid_


CLASS Game:
   PUBLIC:
      CONSTRUCTOR Game(input_file_path, output_file_path):
            OPEN the file at input_file_path as input_file

            DECLARE width as the integer cast of the next word in input_file
            SET board_ to a Board of size width by width
            ITERATE width times with a variable row:
                  ITERATE width times with a variable col:
                        CALL board_.SetCell(kEmptyCell, row, col)

            SET prev_plays_checked_ as the integer cast of the next word in
                              input_file

            SET output_file_path_ to output_file_path

            ITERATE until end of file is reached:
                  DECLARE Int tail_row as the integer cast of the next word
                                    in input_file
                  DECLARE Int tail_col as the integer cast of the next word
                                    in input_file
                  DECLARE Int head_row as the integer cast of the next word
                                    in input_file
                  DECLARE Int head_col as the integer cast of the next word
                                    in input_file

                  APPEND a new ParametricLine(tail_row, tail_col, head_row,
                                    head_col) to plays_

      FUNCTION IsPlayValid(play_num):
            IF prev_plays_checked_ > 0:
                  DECLARE first_play_checked as play_num -
                                    prev_plays_checked_
```

```
            IF first_play_checked < 0:
                  SET first_play_checked to 0

            ITERATE from first_play_checked to just before play_num:
                  IF plays_[first_play_checked].GetMidPoint() is equal
                              to plays_[play_num].GetMidPoint() and
                              the dot product of
                              plays_[first_play_checked].GetVector()
                              and plays_[play_num].GetVector() equals
                              0:
                        RETURN false

      RETURN true

FUNCTION Play():
      CALL board_.DisplayBoard()

      DECLARE curr_color

      ITERATE from 0 to the length of plays_ using curr_play_:
            IF curr_play_ is even:
                  SET curr_color to kBlackCell
            ELSE:
                  SET curr_color to kWhiteCell

            IF IsPlayValid(curr_play_):
                  CALL board_.PlotLine(lines_(curr_play_), curr_color)

            CALL board_.DisplayBoard()

      OPEN output_file at output_file_path_

      DECLARE grid as board_.GetGrid()

      ITERATE grid's size times with a variable row:
            ITERATE grid's size times with a variable col:
                  PRINT grid[row][col] and a space after it

            PRINT a newline character
```

```
          WRITE to output_file something like “Player %s: %d cells; Player
                      %s: %d cells\n”, filling in the %d’s with
                      board_.CountColor(kBlackCell) and
                      board_.CountColor(kWhiteCell) respectively, as well
                      as the %s’s with kBlackCell and kWhiteCell

          CLOSE output_file

  PRIVATE:
     INSTANCE VARIABLE Board board_
     INSTANCE VARIABLE Std::Stack of ParametricLines plays_
     INSTANCE VARIABLE Unsigned Int prev_plays_checked_
     INSTANCE VARIABLE String output_file_path_
```