Implementation and Testing for 2D Raycasting

Lukas Jonca
2020-05-10
ICS4U

Since the main menu was straightforward and easy to implement it was implemented first. The main menu only consists of two options either the line intersection demo or the 2D raycast demo.

Main menu:
On each button click the the button clicked on would simply be outputted on the console.

ActionPerformed	Expected	Returned	Solution
Line Segments	“Line clicked”	Expected	
2D Raycast	“Ray clicked”	Expected	

Line Segment Intersection:
Before creating the 2D Raycast I thought it would be a better idea to instead to understand the math behind it and simply try finding the intersection of lines.

Line Constructor	Expected	Returned	Solution
Initialize variables	Initialized variables	Expected	
Generate Random lines	Random coordinates	Expected	

Determining if the line segments intersect:
By finding the T and U values we can determine if an intersection exists if 0<T<1 and 0<U<1.

FindT	Expected	Returned	Solution
Find T value	T value	Wrong value	Fix equation

FindU	Expected	Returned	Solution
Find U value	U value	Expected	

Finding Intersection Coordinates:

Intersection X	Expected	Returned	Solution
Line Intersection X	X of intersection	Expected	

Intersection Y	Expected	Returned	Solution
Line Intersection Y	Y of intersection	Incorrect	Wrong sign

Drawing the lines:

Paint Component	Expected	Returned	Solution
Draws Lines	Lines Drawn	Expected	
Draws Intersection	Draws Oval at point	Wrong spot	Error in math
Labels Points	Labels 1-4	Expected	

2D Raycasting:
The 2D raycasting follows the same math and logic but no does this with many more lines and instead resets the end point of the line to the coordinate of the intersection.

GUI Constructor	Expected	Returned	Solution
Initialize Variables	Variables initialized	Expected	
Initialize GUI	GUI initialized	Expected	

Menu Bar:
Two menu options are added to the GUIs menu save and load. The Save button saves the player object object to a file using serialization. The load button loads a saved serialized object from a file to the board.

actionPerformed	Expected	Returned	Solution
Save	Saved to file	Error	Implement serialization
Load	Load from file	Expected	

Player Controls:
The demo will need to be interactive so this means the player will need to be able to be controlled. The player will need to be able to move forward/backward and to rotate left and right.

key Pressed	Expected	Returned	Solution
Up arrow	Forward move	Expected	
Down arrow	Backwards move	Expected	
Right arrow	Rotate right	Error out of bounds	Resetting degrees
Left arrow	Rotate left	Error out of bounds	Resetting degrees

Based on the player's movements the rays were also moved and redrawn, the player's FOV was also incorporated and shown by the colors green and red.

Finding Intercepts:
Every time after the player is moved the program must again check where each ray intercepts with walls.

First the program finds the T and U value for each ray with each wall, the same method is used from the line intersection demo.

After if the $0<T<1$ and $0<U<1$ then an intercept exists and the T value can be used to find the X and Y intercept.

$$X\ Intercept = X1 + T(X2 - X1) \qquad Y\ Intercept = Y1 + T(Y2 - Y1)$$

Once the intercept is found the rays X2 and Y2 can be changed to the calculated values.

Draw Board:
Once all intercepts and line coordinates are reset they are redrawn on the screen using the line and wall class variables and for loops.

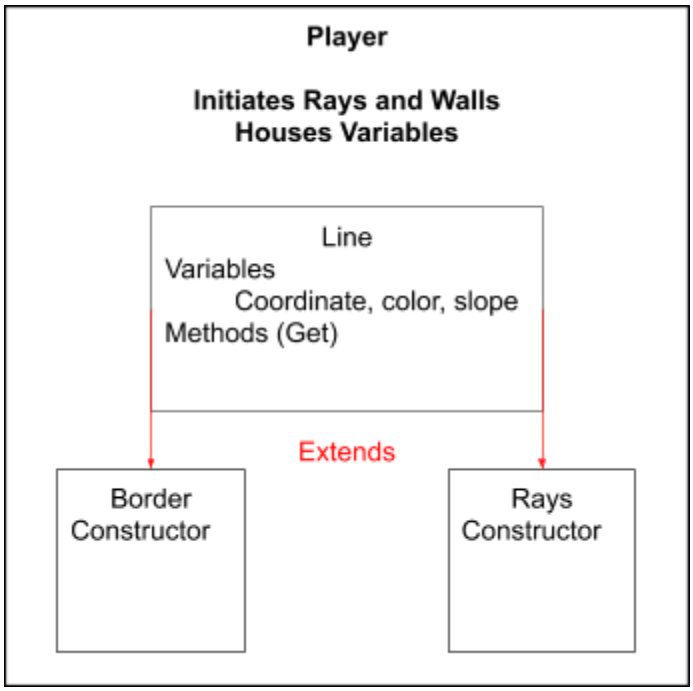
Paint Component	Expected	Returned	Solution
Draw rays	Draws rays	Expected	
Draw walls	Draws walls	Wrong walls	Switched X and Y

Saving/Loading Objects:
Saving each class variable individually would be cumbersome and time consuming so instead entire objects were saved under the player object using serialization.

SavePlayer	Expected	Returned	Solution
Save object to file	Serialized file	Error	Implement serialization

LoadPlayer	Expected	Returned	Solution
Loads player	Load player from file	Expected	

Class Structure:



The line class is the abstract class and the ray class and border class extends from it.

The player class then houses all the lines and borders, but also variables such as the players degree (direction it is facing) and the field of view of the player.

This structure is on purpose so that any other part of the program can easily access these variables. It also allows for all the programs variables to be saved extremely easily.

Line Class:
Both rays and borders (aka walls) can be considered lines, they both consist of 2 X and 2 Y coordinates so their classes will follow similar structures. But they do have some difference mainly being that the walls are stationary objects that do not allow rays to pass through them.

The line class is an abstract class that houses all the methods and variables to be used from by both the ray and border classes.

All the methods within the line class are used to access the variables of the object outside of the class by other parts of the program ie paintComponent or keyPress. The parts of the class are very straightforward and as expected worked without issue.

Line Class	Expected	Returned	Solution
Get X1	Returns X1	Expected	
Get X2	Returns X2	Expected	
Get Y1	Returns Y1	Expected	
Get Y2	Returns Y2	Expected	
Get color	Returns color	Expected	
Get slope	Returns slope	Expected	

Border and Ray Class:
Both classes consisted of only constructor and were used as means of housing variables and distinguishing between the two similar objects.

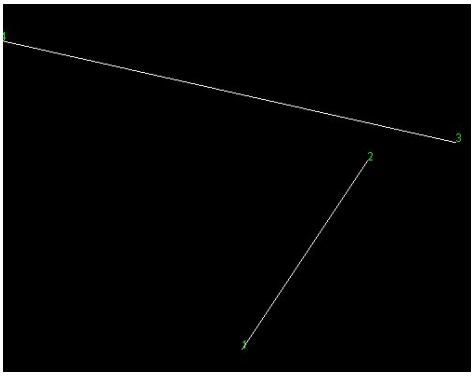
Border Class	Expected	Returned	Solution
Constructor	Set variables	Expected	

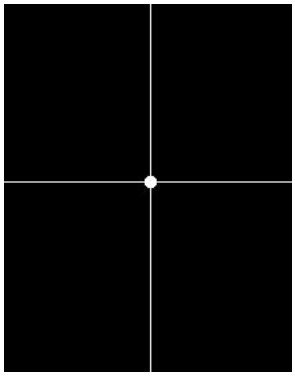
Ray class	Expected	Returned	Solution
Constructor	Set variables	Expected	

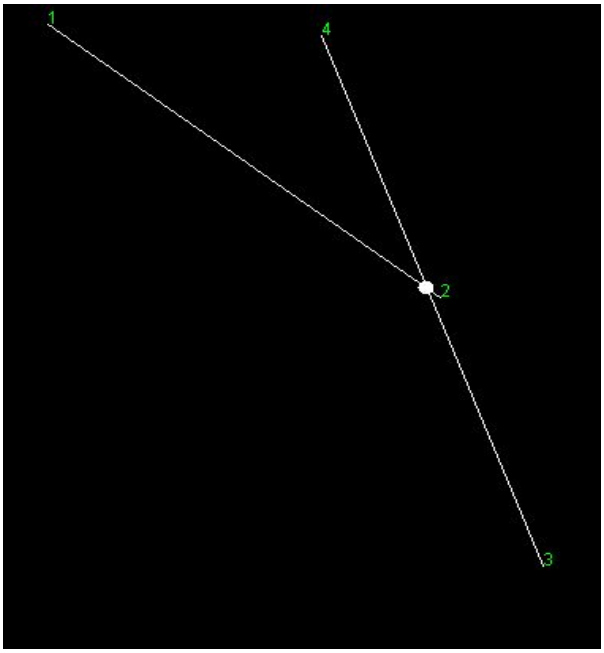
Systems Test:
First the 2 line segment was system tested to ensure intersections could be detected properly before implementing it to the full demo.

2 Line Segment Intersection:

Systems Test:	Expected	Returned	Solution
Intersecting	POI drawn	Expected	
Non Intersecting	No POI	Expected	
Perpendicular	POI drawn	Expected	
Parallel	No POI	Error (dividing by 0)	Implement if statement





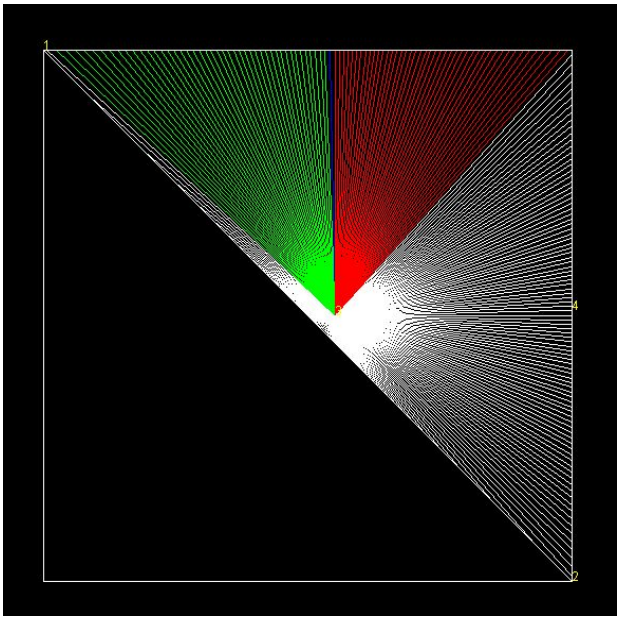


After each type of possible case was tested the program will be robust and function under all circumstances.

2D Raycasting:
The second part of the systems test was the actual 2D raycasting after it has actually been implemented. The equations used previously will be used again but this time on a larger scale with many more lines. There are 4 important cases to be tested the first intersecting and non intersecting then making sure the code works with perfectly vertical and horizontal lines.

Intersections:

System Test:	Expected	Returned	Solution
Intersecting	Collides with wall	Expected	
Non Intersecting	Radiates out	Expected	
Vertical wall	Collides with wall	Expected	
Horizontal Wall	Collides with wall	Expected	



As the image shows the rays successfully colidewith the walls and stop. It also show that the rays automatically collide with the closest boundary.

The blue ray represents the direction the player is actually facing.

The green and red colors represent the players FOV that moves with the direction the player is facing.

The white lines are simply representative of the line of sight.

Player Movement:
Player movement is an important feature ensuring the player can turn and its FOV follows also making sure the player can move in the direction its facing by calculating the speeds x and y and incrementing or decrementing.

System Test:	Expected	Returned	Solution
Player rotation left	Rotates left	Expected	
Player rotation right	Rotates right	Expected	
Rotation past 0	offset counter	Error	Improve logic
Rotation past 360	offset counter	Error	Improve logic
Forward movement	Moves forward	Expected	
Backward movement	Moves backward	Expected	

When the player hit 360 or 0 and tried to keep rotating error began occurring so it was important to offset the counter. To do this a else statement was added if the players degree falls above 360 or below 0.

Pseudo Code

Import java packages
Public main menu class that extends JFrame and Implements ActionLister

The constructor initiates and sets up all variables for the main menu.

Main menu constructor
- Initialize menu components (menuBar, menu, menuItem)
- Setup menu bar
- Display GUI screen

The display method is responsible for displaying actual GUI elements such as buttons, text, headers and etc.

Display GUI method
- Initialize menu variables
- Setup bottom pane
- Setup content pane
- Setup Jpanels (myJpanel, topPanel)

- Setup GUI layouts
- Add text “Raycasting”
- Add bottom pane

- Add “line segment” button
- Add “2D raycasting” button
- Add “load saved” button

- Add bottom pane to content pane
- Validate

The action performed method is called when a button or action is taken on the GUI, then based on which button was clicked specific actions are taken such as starting the demo loading previous, etc.

- actionPerformed method
  - Initialize event variable

  - If event equals line segment
    - launch line segment GUI

```
        If event equals 2D Raycast
            Launch 2D raycast

        If event equals load saved
            Launch 2D raycast
            Load saved file method


    Main method
        Print out launching
        Initialize and set up main menu GUI
```

Line Segment Class: the line segment class is used to generate and find the intersection of two different lines.

```
Import Java Packages
public line segment class extending JFrame implementing ActionListener

Initialize coordinate variables
```

The line segment constructor is responsible for initializing and setting up variables for new GUIs.

```
Line segment constructor
    declare JMenuBar, JMenu, JMenuItem
    Set up GUI board
    Initialize J menu bar
    Generate random line coordinates
```

The Board class contains the paintComponent method which is responsible for creating custom graphics and updating those graphics including lines, points and intersections.

```
public class Board extending JPanel
        paintComponent method
            Declare variables
            Set color to green
            label points of lines
            Set color white
            Draw line 1 and 2
            Find t and u

            if 1>t>0  and 1> u>0
                Find x and y intersections
                Draw POI oval
```

Intersection method, the intersection methods are used to find x and y intersections of the lines. Both the functions take 4 x and 4 y coordinates as input and return a single value.

```
IntersectionX method
        Declare ints
        Calculate numerator = (x1*y2 - y1*x2)*(x3 -x4) - (x1 - x2)*(x3*y4 - y3*x4)
        Calculate denominator = (x1 - x2)*(y3 - y4) - (y1 - y2)*(x3 - x4)
        If denominator = 0
                return 0

        Point x = numerator/denominator
        Return x

IntersectionY method
        Declare ints
        Calculate numerator = (x1*y2 - y1*x2)*(y3 -y4) - (y1 - y2)*(x3*y4-y3*x4)
        Calculate denominator = (x1 - x2)*(y3 - y4) - (y1 - y2)*(x3 - x4)
        If denominator = 0
                return 0

        Point y = numerator/denominator
        Return y
```

To determine if 2 line segments intersect t and u are used if 1>t>0 and 1> u>0 then the line segments are intersecting. The methods for t and u are similar and both take 4 x and 4 y coordinates as input and return a single value.

Find t method

```
numerator = (x1 - x3)*(y3 - y4) - (y1 - y3)*(x3 - x4)
denominator = (x1 - x2)*(y3 - y4) - (y1 - y2)*(x3 - x4)

if denominator = 0
        return 2

t = numerator/denominator

return t
```

Find u method

```
numerator = (x1 - x2)*(y1 - y3) - (y1 -y2)*(x1-x3)
denominator = (x1 - x2)*(y3 - y4) - (y1 - y2)*(x3 - x4)

if denominator = 0
        return 2

u = numerator/denominator

return u
```

2D Raycast GUI:
The GUI class used for visualizing/calculating the 2D ray cast intersections, the class allows for player input and character movement.

Import Java Packages
public class GUI extending JFrame and implementing ActionListener

    initialize player variables
    Initialize coordinate variables

The constructor initiates and sets up all variables for the raycast GUI.

    GUI constructor
        declare JMenuBar, JMenu, JMenuItem
        setup GUI board
        Initialize and setup the JMenuBar
        set up menu bar option file
        set up menu item save option
        set up menu item load option
        Initialize key press variable
        Add key listener

The save method is used to save the entire player object to file using serialization. Input is the player and the output is the file.

    Save player method
        Creates a File output

        try for errors
            set output to file player.ser
            Output player object to file
            Close file
            Print changes saved
        Catch errors
            Print errors

The load method is used to load previous player objects from a file into a player variable. The input is the name of the file and the method outputs the loaded player file.

```
Load player method
    initialize stream and board variables

    Try for errors
        Initialize stream
        Load board
        Close file
    Catch for errors
        Print out errors

    Return loaded player
```

The action performed method is called when a button or action is taken on the GUI, then based on which button was clicked specific actions are taken such as saving or loading a file.

```
actionPerformed method
    get action performed

    if save option pressed
        save player to file using method

    if load option pressed
        load player from file using method
        player = loaded player
```

The press class implements a keylistener to listen for user keyboard inputs.

```
 class Press implementing KeyListener
```

The key pressed method is called whenever a key on keyboard is pressed

```
Key pressed method
    Initialize integer variables

    If right arrow pressed

        Try block
            if player degree is less than 360
                Increment player degree +1
            Else
                Player degree = 0
```

```
                    If player degree > 0
                        Player ret previous ray color

            Catch errors
                    Print errors

If left arrow pressed
        Try block
                    If player degree > 0
                        Decrement player degree
                    Else
                        Player degree = 360

                    If player degree < 360
                        Rest previous ray color

        Catch errors
                    Print errors

If up arrow pressed
        increment coordinates
        Player Y= Y + speed Y
        Player X = X + speed X

If down arrow pressed
        decrement coordinates
        Player Y = Y - speed Y
        Player X  = X - speed X

For loop 0 to 360
        Set ray color at index = white

For loop 0 to FOV/2

        try block
                    If player degree + index< 360
                        set color of ray to green
                    Else
                         set color of ray to red

                    If player - i >0
                         set color of ray to red
                    Else
```

```
                    set color of ray to green

                if player FOV does not go past 0 or 360
                    Ray color set to white

                Set player degree ray to blue
Initialize int variables x, y intersection
Initialize double variable t and u

For loop 0 tp player rays size
        Set ray coordinates x1 and y1 to player

        if i is between 0 and 90
            Set ray x2 at i = 7000
            set  ray y2 at i = raySlope(x2) + playerY
        Set ray x2 at 90 = playerX
        Set ray y2 at 90 = 7000

        if i is between 90 and 180
            Set ray x2 at i = -7000
            set  ray y2 at i = raySlope(x2) + playerY
        Set ray x2 at 90 = -7000
        Set ray y2 at 90 = playerY

        if i is between 180 and 270
            Set ray x2 at i = -7000
            set  ray y2 at i = raySlope(x2) + playerY
        Set ray x2 at 90 = playerX
        Set ray y2 at 90 = -7000

        if i is between 270 and 360
            Set ray x2 at i = 7000
            set  ray y2 at i = raySlope(x2) + playerY

For loop j 0 to player rays size
        For loop i player wall size
            set  coordinates x1,x2,x3,x4,y1,y2,y3,y4
            Find t, u using method
            if 1>t>0 and 1>u>0
                Try block
                    Calculate x,y intersection
                    Set player ray at j to x, y
                Catch and print errors
```

Board class: is positioned within the GUI class, it is used to display custom graphics on the GUI.
Board class implementing JPanel

Paint component method draws graphics onto the board.
Paint component
Declare integer x1,x2,y1,y2

For loop i 0 to player ray size
Set color ray at i color
Get x1,x2,y1,y22 from ray at i
Draw line from (x1,y1,x2,y2)

Set color white

For int i 0 to player walls size
Draw player wall (wall(i)X1, wall(i)Y1, wall(i)X2, wall(i)Y2)

Player Class: the player class is used to house all major variables of the program things such as coordinates, degrees and FOV. The class is also important as it initially sets up the rays and generates the walls.

Import java packages

Public class player implementing serialization

Initialize Int playerX, playerY, Int width
Initialize walls array list
Initialize rays array list
Initialize degree = 0
Initialize FOV = 90

The player constructor method is responsible for initially setting up and generating the rays and walls.
Player Constructor method
set player coordinates player x = 350,player y = 350
initialize variables double rad and m
initialize variable integer lineY2

For i 0 to 90

```
            Rads =  (i to radians)
            Slope = tan(rad)
            LineY2 =  slope*7000+playerY

            Add new ray to list (player x, player y, 7000, lineY2, m, white)
        Add new ray to list (player x, player y, 7000, 0, 0, white)

        For i 91 to 180
            Rads =  (i to radians)
            Slope = tan(rad)
            LineY2 =  slope*-7000+playerY

            Add new ray to list (player x, player y, -7000, lineY2, m, white)

        For i 180 to 90
            Rads =  (i to radians)
            Slope = tan(rad)
            LineY2 =  slope*7000+playerY

            Add new ray to list (player x, player y, -7000, lineY2, m, white)
        Add new ray to list (player x, player y, -7000, 0, 0, white)

        For i 271 to 360
            Rads =  (i to radians)
            Slope = tan(rad)
            LineY2 =  slope*7000+playerY

            Add new ray to list (player x, player y, 7000, lineY2, m, white)

        For i 0 to 10
            Generate randomX1, randomY1, randomX2, randomY2
            Add wall(randomX1, randomY1, randomX2, randomY2)


method for getting playerX
        Return playerX

method for getting playerY
        Return playerY

method for getting rays (input i)
        Return player rays at i
```