3802ICT - Modelling and Visualisation - Assignment 1

Student: Lochie Ashcroft - s5080439

Programming Language: Typescript

Development Environment

Editor: Visual Studio Code

Linter: TSLint (recommended configuration)

Compiler: tsc

Build script: `Makefile`

Docs: TypeDoc HTML and with the Markdown plugin

Directory Structure

Directory	Purpose
docs	documentation
images	images for `index.html`
js	output of tsc
sounds	game sounds
ts	source code

Using a makefile as this is not a node module

```
all:
    tsc ./ts/*.ts --outDir ./js/ && browserify ./js/*.js -o game.js

clean:
    rm -rf ./js/*
```

- The Typescript source is located at `/ts/`
- The compiled Typescript is output to `/js/`
- Browserify is then used to pack the contents of `/js/` into a single file called `game.js`
  - Browserify is used so in `index.html` only one javascript file needs to be sourced.
- **Optional** uglify-js is ran to compress `game.js`

Libraries that the project is based on

HTML5 canvas API

Game

In the classic Asteroids game, you steer a ship in space, clearing it of asteroids. Asteroids are destroyed by shooting them, and if your ship collides with an asteroid, it is destroyed. Imaginary spaceship construction has improved since the 80s. Now the hulls can withstand a collision and it is the easiest way to destroy the asteroids. Your spaceship will be drawn with several filled polygons, at least one of which should be concave. The spaceship must only accelerate in the direction it is pointing,so it must be able to rotate. Control the spaceship with keys. Asteroids should bounce off the boundaries of the display. New asteroids should spawn at regular or random intervals, with 0 to many on screen at the same time. How the game is scored is up to you, but a score must be visible (in the console as a last resort). At least collisions should cause a sound effect.

The completed game can be found on my website http://lochieashcroft.com/asteroid/

Game play

In the game Rameroids the player controls a space ship and the only goal is to collide with asteroids. There is a maximum number of asteroids that can be on the screen at any given time and when the current number drops below this there is a random probability that an asteroid will spawn.

While playing the game the user can change the configuration.

- Toggle the view of the bounding boxes (hit boxes)
- Toggle the display of an FPS counter
- The game resolution
- The maximum number of asteroids ont he screen at once

Controls

Key	Action
w	thrust
a	rotate left
d	rotate right

Scoring

Points are earned by colliding into asteroids, 1 point per asteroid.

Graphics Engine Documentation

Below is the output of Typedoc with the markdown plugin. For a better experience open `./docs/index.html` or go to my website http://lochieashcroft.com/asteroid/docs/ for the HTML version. **Lochie's Graphics Engine**

Globals

Index

External modules

- "Colour"
- "IPoint"
- "LGE"
- "Matrix"
- "Pixel"
- "Polygon"
- "Rectangle"
- "ShapeFactory"
- "Utils"

External modules

"Colour"

- **"Colour"**:

Defined in Colour.ts:1

Colour

- **Colour**:

Defined in Colour.ts:15

Colour

Represents a colour (red, green, blue, alpha).

A wrapper over the html rgba format

Example

```
const red = new Colour(255, 0, 0, 100);
console.log(red.toString());
```

constructor

+ **new Colour**(`r`: number, `g`: number, `b`: number, `a`: number): *Colour*

Defined in Colour.ts:19

Parameters:

Name	Type	Description
`r`	number	red (0-255)
`g`	number	green (0-255)
`b`	number	blue (0 - 255)
`a`	number	alpha (0 - 100)

Returns: *Colour*

a

- **a**: *number*

Defined in Colour.ts:19

b

- **b**: *number*

Defined in Colour.ts:18

g

- **g**: *number*

Defined in Colour.ts:17

r

- **r**: *number*

Defined in Colour.ts:16

toString

toString(): *string*

Defined in Colour.ts:41

string representation

Returns: *string*

returns string format 'rgba(r, g, b, a)'

Colours: ***object***

Defined in Colour.ts:50

Colours

A library of colours

- **black**: *Colour* = new Colour(0, 0, 0, 100)
- **blue**: *Colour* = new Colour(0, 0, 255, 100)
- **brown1**: *Colour* = new Colour(135, 54, 0, 100)
- **brown2**: *Colour* = new Colour(110, 44, 0, 100)
- **brown3**: *Colour* = new Colour(93, 64, 55, 100)
- **green**: *Colour* = new Colour(0, 255, 0, 100)
- **orange**: *Colour* = new Colour(243, 156, 18, 100)
- **red**: *Colour* = new Colour(255, 0, 0, 100)
- **silver**: *Colour* = new Colour(192, 192, 192, 100)
- **white**: *Colour* = new Colour(255, 255, 255, 100)

“IPoint”

- **“IPoint”**:

Defined in IPoint.ts:1

IPoint

- **IPoint**:

Defined in IPoint.ts:12

IPoint

An interface which represents a point on the canvas (cartesian plane)

Example

```
const startPoint: IPoint = {x: 100, y: 100};
```

x

- **x**: *number*

Defined in IPoint.ts:13

y

- **y**: *number*

Defined in IPoint.ts:14

“LGE”

- **“LGE”**:

Defined in LGE.ts:1

LGE

- **LGE**:

Defined in LGE.ts:96

Lochie's Graphics Engine

A 2D graphics engine which uses the HTML5 canvas.

Components

Lochie's Graphics Engine (LGE) is comprised of multiple components this is the main class which handles rendering.

See IPoint, Polygon Rectangle, Pixel, Utils, ShapeFactory, Colour for more details.

Coordinate Spaces

The origin (0, 0) is the top left of the canvas, when drawing coordinates are taken as is - there is no manipulation. The resolution is determined from the canvas in the constructor.

Pixel size

The pixel size determines the size of each pixel but does not modify the coordinate system.

Rendering process

LGE primarily deals with polygons drawPolygon(poly: Polygon, colour?: Colour) will be used as an example.

1. the points are extracted from the polygon
2. if the polygon has a fill colour the scan line fill algorithm is used to draw and fill the polygon
3. if a colour was provided when calling the function that will be the outline colour else the polygon's colour will be used
4. the outline of the polygon is drawn using the DDA algorithm

The polygon will now stay on the canvas until the canvas has been cleared.

Rendering example

The following simple example will render two squares on the canvas and rotate them. The drawPolygonBuffer(buffer: Polygon[] function is used which iterates over the buffer and calls drawPolygon(poly: Polygon, colour?: Colour)

Assuming: ctx variable is the CanvasRenderingContext2D

```
// instantiate LGE
const lge: LGE = new LGE(ctx, 1, 'scanLine')

// degrees / second
const rotationSpeed = 360;

// polygons
const square1: Polygon = sf.square(100, 100, 100, 100);
const square2: Polygon = sf.square(400, 400, 100, 100);

// frame rendering time deltas
let frameTimeDelta = 0;
let lastFrameTime = 0;

// the draw function
const draw = (timestamp: number) => {

   // clear the canvas
   lge.clear();

   // calculate time delta
   frameTimeDelta = (timestamp - lastFrameTime) / 1000;
   lastFrameTime = timestamp;

   // rotate the squares accurately (taking previous frame render time into account)
   square1.rotate(rotationSpeed * frameTimeDelta);
   square2.rotate(rotationSpeed * frameTimeDelta);

   // create the buffer of polygons to draw
   const polygonBuffer: Polygon[] = [square1, square2];

   // draw every polygon in the buffer
   lge.drawPolygonBuffer(polygonBuffer);

   // call this function on the next animation frame
   requestAnimationFrame(draw);
}
```

```
// start the draw loop
requestAnimationFrame(draw);
```

constructor

+ **new LGE**(`ctx`: CanvasRenderingContext2D, `PIXEL_SIZE`: number, `fillMethod`: string | null): *LGE*

Defined in LGE.ts:103

Parameters:

Name	Type	Description
`ctx`	CanvasRenderingContext2D	canvas ctx
`PIXEL_SIZE`	number	size of each pixel
`fillMethod`	string | null	'scanLine'/null/'other'

Returns: *LGE*

`Private` PIXEL_SIZE

• **PIXEL_SIZE**: *number*

Defined in LGE.ts:98

`Private` ctx

• **ctx**: *CanvasRenderingContext2D*

Defined in LGE.ts:103

`Private` fillMethod

• **fillMethod**: *string | null*

Defined in LGE.ts:99

resolution

• **resolution**: *IPoint*

Defined in LGE.ts:97

`Private` rotationMatrix

- **rotationMatrix**: *Matrix*

Defined in LGE.ts:101

`Private` transformationMatrices

- **transformationMatrices**: *Matrix[]* = []

Defined in LGE.ts:102

`Private` translationMatrix

- **translationMatrix**: *Matrix*

Defined in LGE.ts:100

addTranslations

addTranslations(`translationMatrix`: any): *void*

Defined in LGE.ts:517

Parameters:

Name	Type
`translationMatrix`	any

Returns: *void*

clear

clear(): *void*

Defined in LGE.ts:487

Clears the whole canvas

Returns: *void*

clearSmart

clearSmart(`polygons`: Polygon[]): *void*

Defined in LGE.ts:496

Clears the canvas by clearing the polygon's bounding boxes

Parameters:

Name	Type	Description
`polygons`	Polygon[]	polygons to be cleared

Returns: *void*

drawCircle

drawCircle(`xc`: number, `yc`: number, `radius`: number, `samples`: number, `colour`: Colour): *void*

Defined in LGE.ts:426

Draws a circle

`remarks` doesnt work

Parameters:

Name	Type	Description
`xc`	number	x pos (centre)
`yc`	number	y pos (centre)
`radius`	number	radius
`samples`	number	number of samples
`colour`	Colour	Colour

Returns: *void*

drawLine

drawLine(`start`: IPoint, `end`: IPoint, `colour`: Colour): *void*

Defined in LGE.ts:181

Draws a coloured line between two IPoints using the DDA algorithm where each pixel is the size of PIXEL_SIZE

Parameters:

Name	Type	Description
`start`	IPoint	first IPoint

Name	Type	Description
`end`	IPoint	end IPoint
`colour`	Colour	Colour

Returns: *void*

drawPath

drawPath(`points`: IPoint[], `colour`: Colour): *void*

Defined in LGE.ts:219

Draws a path between a series of IPoints using drawLine();

Parameters:

Name	Type	Description
`points`	IPoint[]	IPoint[]
`colour`	Colour	Colour

Returns: *void*

drawPolygon

drawPolygon(`poly`: Polygon, `colour?`: Colour): *void*

Defined in LGE.ts:316

Fills the polygon using the Polygons fill colour and draws the outline with colour or the Polygons colour

Parameters:

Name	Type	Description
`poly`	Polygon	Polygon
`colour?`	Colour	Optional Colour - overwrites polygon colour

Returns: *void*

drawPolygonBuffer

drawPolygonBuffer(`buffer`: Polygon[]): *void*

Defined in LGE.ts:337

Draws an array of Polygons using drawPolygon()

Parameters:

Name	Type	Description
`buffer`	Polygon[]	Polygon[]

Returns: *void*

drawRectangle

drawRectangle(`x`: number, `y`: number, `width`: number, `height`: number, `colour`: Colour): *void*

Defined in LGE.ts:372

Draws a rectangle

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos
`width`	number	width
`height`	number	height
`colour`	Colour	Colour

Returns: *void*

drawTriangle

drawTriangle(`points`: IPoint[], `colour`: Colour): *void*

Defined in LGE.ts:349

Draws a triangle from 3 points

Parameters:

Name	Type	Description
`points`	IPoint[]	Points
`colour`	Colour	Colour

Returns: *void*

fillCircle

fillCircle(`x`: number, `y`: number, `radius`: number, `samples`: number, `colour`: Colour): *void*

Defined in LGE.ts:464

Fills a circle

`remarks` doesnt work

Parameters:

Name	Type	Description
`x`	number	-
`y`	number	-
`radius`	number	radius
`samples`	number	number of samples
`colour`	Colour	Colour

Returns: *void*

fillPolygon

fillPolygon(`poly`: Polygon, `colour`: Colour): *void*

Defined in LGE.ts:295

Fills a polygon based on the method decided on at initialization

Parameters:

Name	Type	Description
`poly`	Polygon	Polygon
`colour`	Colour	Colour

Returns: *void*

fillRectangle

fillRectangle(`x`: number, `y`: number, `width`: number, `height`: number, `colour`: Colour): *void*

Defined in LGE.ts:398

Fills a rectangle

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos
`width`	number	width
`height`	number	height
`colour`	Colour	Colour

Returns: *void*

fillTriangle

fillTriangle(`points`: IPoint[], `colour`: Colour): *void*

Defined in LGE.ts:359

Fills a triangle from 3 points

Parameters:

Name	Type	Description
`points`	IPoint[]	IPoint[]
`colour`	Colour	Colour

Returns: *void*

popTranslation

popTranslation(`count`: number | undefined): *void*

Defined in LGE.ts:521

Parameters:

Name	Type
`count`	number | undefined

Returns: *void*

scanLineFill

scanLineFill(`poly`: Polygon, `colour`: Colour): *void*

Defined in LGE.ts:231

Fills a Polygon using the scanLineFill algorithm with the colour provided

Parameters:

Name	Type	Description
`poly`	Polygon	Polygon
`colour`	Colour	Colour

Returns: *void*

setRotation

setRotation(`angle`: number): *void*

Defined in LGE.ts:153

updates rotation matrix and applies to transformation matrix

Parameters:

Name	Type	Description
`angle`	number	radians

Returns: *void*

setTranslation

setTranslation(`dX`: number, `dY`: number): *void*

Defined in LGE.ts:168

sets translation matrix values and updates the transformation matrix

Parameters:

Name	Type	Description
`dX`	number	change in x
`dY`	number	change in y

Returns: *void*

updateTransformationMatrix

updateTransformationMatrix(): *Matrix*

Defined in LGE.ts:132

combines the translation matrix and rotation matrix into a single matrix and adds to stack

Returns: *Matrix*

"Matrix"

- **"Matrix"**:

Defined in Matrix.ts:1

Matrix

- **Matrix**:

Defined in Matrix.ts:18

Matrix

The Matrix class is used for matrix operations most commonly for translating and rotating polygons which is performed using transformation matrices. If no values are provided in the constructor the values are the identity matrix.

Example

```
const identityMatrix: Matrix: = new Matrix(null);
const otherMatrix: Matrix = new Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]]);

console.log(identityMatrix.multiply(otherMatrix));
```

constructor

+ **new Matrix**(`values`: number[][] | null): *Matrix*

Defined in Matrix.ts:23

Parameters:

Name	Type
`values`	number[][] | null

Returns: *Matrix*

height

- **height**: *number*

Defined in Matrix.ts:21

identityMatrix

- **identityMatrix**: *number[][]* = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

Defined in Matrix.ts:23

values

- **values**: *number[][]*

Defined in Matrix.ts:19

width

- **width**: *number*

Defined in Matrix.ts:20

add

add(`b`: Matrix): *Matrix*

Defined in Matrix.ts:41

Adds two matrices and returns the result

Parameters:

Name	Type	Description
b	Matrix	Matrix

Returns: *Matrix*

Matrix

multiply

multiply(b: Matrix): *Matrix*

Defined in Matrix.ts:64

Multiples by Matrix b and returns the result

Parameters:

Name	Type	Description
b	Matrix	Matrix

Returns: *Matrix*

Matrix

zero

zero(): *void*

Defined in Matrix.ts:90

Zeros the matrix values

Returns: *void*

"Pixel"

- **"Pixel"**:

Defined in Pixel.ts:1

Pixel

- **Pixel**:

Defined in Pixel.ts:20

Pixel

Represents a 'pixel'.

The pixel size is provided when instantiating LGE.

When a pixel is drawn it creates a rectangle and uses the HTML5 canvas API to draw this rectangle. This is the only part of the engine that uses the HTML5 canvas API directly to draw.

Example

```
new Pixel(100, 100, 4, ctx, new Colour(255, 0, 0, 100));
```

constructor

+ **new Pixel**(`x`: number, `y`: number, `size`: number, `ctx`: CanvasRenderingContext2D, `colour`: Colour): *Pixel*

Defined in Pixel.ts:23

Draws a 'pixel' on the canvas

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos
`size`	number	size
`ctx`	CanvasRenderingContext2D	canvas context
`colour`	Colour	colour

Returns: *Pixel*

colour

- **colour**: *Colour*

Defined in Pixel.ts:21

`Private ctx`

- **ctx**: *CanvasRenderingContext2D*

Defined in Pixel.ts:22

`Private rec`

- **rec**: *Rectangle*

Defined in Pixel.ts:23

“Polygon”

- **“Polygon”**:

Defined in Polygon.ts:1

Polygon

- **Polygon**:

Defined in Polygon.ts:52

Polygon

Represents a polygon by an array of IPoints.

The polygon is the most important component for LGE.

Each polygon keeps track of its centrepoint and bounding box if specified.

The polygon stores the colour, fill colour and provides functions to translate and rotate. The angle of the polygon is also stored for convenience.

the decompose function decomposes the polygon into triangles and stored in the triangles property.

Note: when translating and rotating the bounding box will be updated if present

Note: The points are absolute and transform/rotate functions modify the points in place

Example

```
// define the points
const points: IPoint[] = [
 {x: 100, y: 100},
 {x: 200, y: 100},
 {x: 150, y: 200}
];

// create the triangle defined by the points
const triangle: Polygon = new Polygon(points);

// update the fill colour of the polygon to be red
triangle.fillColour = new Colour(255, 0, 0, 100);

// output the points
console.log(triangle.points);

// translate the polygon
triangle.translate(100, 100);

// output the points
console.log(triangle.points);
```

constructor

+ **new Polygon**(`points`: IPoint[], `hasBoundingBox?`: boolean): *Polygon*

Defined in Polygon.ts:61

Parameters:

Name	Type	Description
`points`	IPoint[]	points that represent the polygon
`hasBoundingBox?`	boolean	-

Returns: *Polygon*

angle

• **angle**: *number* = 0

Defined in Polygon.ts:57

boundingBox

- **boundingBox**: *Polygon*

Defined in Polygon.ts:61

centrePoint

- **centrePoint**: *IPoint*

Defined in Polygon.ts:60

colour

- **colour**: *Colour*

Defined in Polygon.ts:58

fillColour

- **fillColour**: *Colour | null* = null

Defined in Polygon.ts:59

points

- **points**: *IPoint[]*

Defined in Polygon.ts:53

scale

- **scale**: *number*

Defined in Polygon.ts:56

transformationMatrix

- **transformationMatrix**: *Matrix* = new Matrix(null)

Defined in Polygon.ts:55

triangles

- **triangles**: *Polygon[]*

Defined in Polygon.ts:54

decompose

decompose(): *Polygon[]*

Defined in Polygon.ts:87

decomposes the polygon into triangles. returns the triangles but also updates the triangles property

Returns: *Polygon[]*

Polygon[]

moveTo

moveTo(`x`: number, `y`: number): *void*

Defined in Polygon.ts:167

Moves the centrepoint of the polygon to (x, y)

`remarks` not recommended

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos

Returns: *void*

rotate

rotate(`angle`: number, `point?`: IPoint): *void*

Defined in Polygon.ts:192

Rotate the polygon by angle degrees. If no point is supplied the rotation point is the centrepoint of the polygon

Parameters:

Name	Type	Description
`angle`	number	degrees
`point?`	IPoint	optional point to rotate around

Returns: *void*

translate

translate(`deltaX`: number, `deltaY`: number): *void*

Defined in Polygon.ts:134

Translate position by deltaX, deltaY

Parameters:

Name	Type	Description
`deltaX`	number	change in x
`deltaY`	number	change in y

Returns: *void*

Polygon

updateBoundingBox

updateBoundingBox(): *void*

Defined in Polygon.ts:242

Update the bounding box of the polygon

Returns: *void*

"Rectangle"

- **"Rectangle"**:

Defined in Rectangle.ts:1

Rectangle

- **Rectangle**:

Defined in Rectangle.ts:21

Rectangle

The Rectangle is drawn using the HTML5 canvas API and its main purpose is to be used by the Pixel class.

Example

```
// instantiate the rectangle
const rect: Rectangle = new Rectangle(100, 100, ctx, 200, 50, new Colour(255, 0, 0, 100));

// draw the rectangle
// ctx.fillStyle = this.fill;
// ctx.fillRect(this.x, this.y, this.width, this.height);
rect.draw();
```

constructor

+ **new Rectangle**(`x`: number, `y`: number, `ctx`: CanvasRenderingContext2D, `width`: number, `height`: number, `colour`: Colour): *Rectangle*

Defined in Rectangle.ts:27

Parameters:

Name	Type	Description
`x`	number	x position
`y`	number	y position
`ctx`	CanvasRenderingContext2D	canvas context
`width`	number	width
`height`	number	height
`colour`	Colour	colour

Returns: *Rectangle*

ctx

- **ctx**: *CanvasRenderingContext2D*

Defined in Rectangle.ts:24

fill

- **fill**: *string*

Defined in Rectangle.ts:27

height

- **height**: *number*

Defined in Rectangle.ts:26

width

- **width**: *number*

Defined in Rectangle.ts:25

x

- **x**: *number*

Defined in Rectangle.ts:22

y

- **y**: *number*

Defined in Rectangle.ts:23

draw

draw(): *void*

Defined in Rectangle.ts:59

draws to the canvas

Returns: *void*

“ShapeFactory”

- **“ShapeFactory”**:

Defined in ShapeFactory.ts:1

ShapeFactory

- **ShapeFactory**:

Defined in ShapeFactory.ts:16

ShapeFactory

provides the ability to generate squares and random polygons using the Factory design pattern.

Example

```
const square: Polygon = ShapeFactory.square(100, 100, 100, 100);
const randomPolygon: Polygon = ShapeFactory.polygon(400, 400, 50, 50, 20);
```

`Static` polygon

polygon(`x`: number, `y`: number, `width`: number, `height`: number, `nPoints`: number): *Polygon*

Defined in ShapeFactory.ts:56

Creates a polygon with n points

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos
`width`	number	width
`height`	number	height
`nPoints`	number	number of points

Returns: *Polygon*

Polygon

`Static` square

square(`x`: number, `y`: number, `width`: number, `height`: number, `boundingBox?`: boolean): *Polygon*

Defined in ShapeFactory.ts:27

Creates a square polygon

Parameters:

Name	Type	Description
`x`	number	x pos
`y`	number	y pos
`width`	number	width
`height`	number	height
`boundingBox?`	boolean	-

Returns: *Polygon*

Polygon

"Utils"

- **"Utils"**:

Defined in Utils.ts:1

Utils

- **Utils**:

Defined in Utils.ts:10

Utils

Utility Class used throughout LGE and the game engine.

`Static` calculateBoundingBox

calculateBoundingBox(`points`: IPoint[]): *IPoint[]*

Defined in Utils.ts:66

Calculates a rectangle bounding box from the supplied points

Parameters:

Name	Type	Description
`points`	IPoint[]	IPoint[]

Returns: *IPoint[]*

IPoint[]

`Static` calculateCentrePoint

calculateCentrePoint(`points`: IPoint[]): *IPoint*

Defined in Utils.ts:102

Calculates the centre point from a series of points from the bounding box

Parameters:

Name	Type	Description
`points`	IPoint[]	IPoint[]

Returns: *IPoint*

IPoint

`Static` insideTriangle

insideTriangle(`p`: IPoint, `t`: Polygon): *boolean*

Defined in Utils.ts:50

Is a IPoint in a triangle

Parameters:

Name	Type	Description
`p`	IPoint	IPoint
`t`	Polygon	triangle

Returns: *boolean*

boolean

`Static` randomInt

randomInt(`max`: number): *number*

Defined in Utils.ts:16

Generates a random number [0…max]

Parameters:

Name	Type	Description
`max`	number	max number

Returns: *number*

random number between 0 and max

`Static` sameSide

sameSide(`a`: IPoint, `b`: IPoint, `l1`: IPoint, `l2`: IPoint): *boolean*

Defined in Utils.ts:30

Are the points on the same side

Parameters:

Name	Type	Description
`a`	IPoint	IPoint
`b`	IPoint	IPoint
`l1`	IPoint	line1
`l2`	IPoint	line2

Returns: *boolean*

boolean