Line.cs

This class represents a line and the power transmission quantities associated with one.
It is derived from the class `Sprite` which defines loading content and positioning.

Class Fields

Name	Type	Explanation
linesToggled	`bool`	This boolean controls whether the line will show a string with the current power flow.
overCapacity	`bool`	A boolean that controls the blinking of the line if it is over capacity.
blinkCounter	`float`	A counter that controls the duration of the blinking when the line is overloaded.
transparencyAlpha	`float`	A value that controls the blinking.
arrow	`Texture2D`	The arrow texture on the line
start	`int`	It is the ID of the bus at the starting point of the line.
end	`int`	It is the ID of the bus at the ending point of the line.
startPos	`Vector2`	The position of the bus at the starting point of the line.
endPos	`Vector2`	The position of the bus at the ending point of the line.
p1	`double`	Powerflow from startPos -> endPos
p2	`double`	Powerflow from endPos -> startPos
pMax	`double`	The maximal power of the line.
r	`double`	The resistance of the line.
x	`double`	The reactance of the line.
id	`int`	The ID of the line.

Class constructor

```
public Line(int id, int start, Vector2 startPos, int end, Vector2
endPos, double pMax, double R, double X)
```

```
{
    this.start = start;
    this.startPos = startPos + new Vector2(30, 30);
    this.end = end;
    this.endPos = endPos  + new Vector2(30, 30);
    this.PMax = pMax;
    this.P1 = 0;
    this.P2 = 0;
    this.R = R;
    this.X = X;
    this.ID = id;
    this.overCapacity = false;
    this.transparencyAlpha = 0f;

    linesToggled = true;
    blinkCounter = 0;
}
```

Input Arguments

- `id`: The Id of the line.
- `start`: The bus ID of the starting point of the line.
- `startPos`: Position of the starting bus.
- `end`: The bus ID of the ending point of the line.
- `endPos`: Position of the ending bus.
- `pMax`: The maximal power of the line.
- `R`: The serial resistance of the line.
- `X`: The serial reactance of the line.

Description

The constructor is very basic and assigns all input values to their respective fields.
Furthermore, it sets `linesToggled` to `true` and `blinkCounter` to 0.

Properties

- Arrow
- Start
- End
- P1
- P2
- PMax

- R
- X
- ID

Arrow

```
public Texture2D Arrow
```

Set

Sets its value to arrow.

Get

Returns the value of arrow.

Start

```
 public int Start
```

Get

Returns the value of start.

End

```
public int End
```

Get

Returns the value of end.

P1

```
public double P1
```

Set

Sets its value to p1.

Get

Returns the value of p1.

P2

```
public double P2
```

Set

Sets its value to p2.

Get

Returns the value of p2.

PMax

```
public double PMax
```

Set

Sets its value to pMax.

Get

Returns the value of pMax.

R

```
public double R
```

Set

Sets its value to 0 if the input value is smaller than 0. Sets its value to r in all other cases.

Get

Returns the value of r.

X

```
public double X
```

Set

Sets its value to 0 if the input value is smaller than 0. Sets its value to x in all other cases.

Get

Returns the value of x.

ID

```
public int ID
```

Set

Sets its value to id.

Get

Returns the value of id.

Methods

```
public String toString()
public void toggleText()
public void Update()
public void Draw(SpriteBatch spriteBatch, SpriteFont theText)
```

toString

```
public String toString()
```

Input Arguments

None.

Description

This method assembles a string of relevant information about the line (namely the current P along with the maximal P of the line).

Returns

A string containing information about the line.

---

toggleText

```
public void toggleText()
```

Input Arguments

None.

Description

This is called when the player toggles the line text with 'T' on the game screen. It makes the text appear or disappear.

Update

```
public void Update()
```

Input Arguments

None.

Description

The update function handles the changing of color for the blinking effect if the line is over capacity.
Once P1 / PMax is greater than 1, it triggers a counter that changes the line color each 30 calls of `Update()`.

Draw

```
public void Draw(SpriteBatch spriteBatch, SpriteFont theText)
```

Input Arguments

- `spriteBatch`: The SpriteBatch used to draw the line.
- `theText`: The SpriteFont used to draw the text.

Description

This method draws a line from startPos to endPos.
First it calculates the distance vector `edge` from the starting position to the end position, then gets the angle from the x-Axis via

```
float angle = (float)Math.Atan2(edge.Y, edge.X);
```

Then it calculates the point in the middle of the connecting line with

```
Vector2 middle = startPos + ((endPos - startPos) / 2);
```

And finally it uses this line to calculate the absolute center of the line since it is 3 pixels wide and otherwise wouldn't be centered

```
Vector2 midNorm = 1.9f * Vector2.Normalize(middle);
midNorm = new Vector2(-midNorm.Y, midNorm.X);
```

`midNorm` is a vector that moves x and y either 0 or 1 pixel depending on the rotation of the line. This is then added to `middle` in the drawing call.
If `P1 < 0` applies the arrow is flipped vertically to show the current flowing in the other direction.

The scale of the arrow is calculated by comparing P1 to PMax.

The color of the line is decided in the same fashion, by generating a value between 0 and 1 depending on the usage.

`Color.Lerp(...)` generates a blended color depending on two colors and a blending value between 0 and 1.

The method draws the line by stretching the 1x1 px line texture to the length of the distance vector and a width of 3 pixels, then rotating it
by `angle` to point it in the direction of the endPosition.

And if the power flow is not zero it will draw the arrow depending on the powerflow and usage.