# DesViz Documentation

Paul Corry

Queensland University of Technology

8/8/2023

## Contents

# 1 Overview

*DesViz* is a collection of Python classes and functions facilitating asynchronous animation for discrete event simulation (DES) models. It is built on top of the *Pyglet* package which provides the underlying graphics functionality. DesViz allows a DES model to write a csv file which is later interpreted by DesViz to configure and move sprites representing background and foreground objects in the simulation. Each line of the csv file gives the simulation time, an animation instruction and set of arguments relating to that instruction. These instructions provide a compact method to specify sprite appearance and movements in ways that are useful in a DES context. Some of the functionality provided by DesViz includes:

- point to point sprite movement with automatic sprite orientation
- sprite movement along predefined poly-line paths
- attachment of slave sprites to a master sprite for coordinated movement
- display of progress bars on the background
- attachment of progress bars to sprites in the foreground
- display of text labels on the background

Along with DesViz, is a companion Excel macro-driven spreadsheet, providing a helpful tool-set for drawing and defining movement paths on a background image. It writes a csv file which defines named paths (sequences of point-to-point movements) over which sprites (i.e. simulation objects) can be moved in the animation.

DesViz has been developed by a DES developer/analyst with no formal training in software development. The priority was on developing DesViz as a functional minimum viable product, which will be incrementally improved and refined over time. There is much scope for improving the design and expanding functionality, and there are currently no bells and whistles such as error handling. Most errors that occur within DesViz (most likely due to input data problems) should be relatively straight forward to debug for any reasonable Python programmer.

# 2 DesViz Animation Setup

DesViz has the following dependencies, and assumes that these packages have already been installed:

- Pyglet
- Pandas

There are two main steps in any DesViz project:

1. Simulation phase: generation of an animation csv file from within a DES model
2. Animation phase: running the DesViz app to render the animation

## 2.1 Simulation Phase

There is no need to import DesViz into the simulation model, and there is no requirement for the simulation to be implemented in Python. All that is required is to output an animation csv file in the required format as specified later in section 3. If however, the simulation model is implemented in Python, DesViz provides some helper functions which can be accessed by importing *DesVizScript.py*. The following function and method are available for setting up and writing the csv file:

- *create_new_DesViz_script_file*(fname)
  - Opens a file with filename *fname* (Type: string) for writing the animation script and writes the required header row, returning a reference to the opened file (Type: file).
- *write_DesViz_command*(fscript, time, command, arg_list)
  - Writes one line of data to the animation file *fscript*.
  - arguments:
    - **fscript**: reference to the file being written to (Type: file)

* **time**: current simulation clock (Type: float)
* **command**: DesViz animation instruction (Type: string)
* **arg_list**: list of arguments applying to the animation instruction (Type: list of mixed types - int, float or string)

Figure 1 shows examples of using the helper function for writing an animation instruction. Section 3 provides detailed descriptions of the various commands and their associated arguments.

```
write_DesViz_command(self.fanim, time, 'image', [truck.name,'resources/truckF.png'])
write_DesViz_command(self.fanim, time, 'move_on', [truck.name,'S'+str(truck.shovel_num)+'toPq',duration,1,0,1])
```

Figure 1: Example calls to *write_DesViz_command* (requires *from DesVisScript import **).

### 2.2 Animation Phase

Figure 2 shows an example program for running a DesViz animation after importing *DesViz.py*.

```
1  import DesViz
2
3  dv = DesViz.DesVizMaster(1500, 750)
4  dv.set_paths('resources/Paths.csv')
5  dv.set_script('pmc_script.csv')
6  dv.set_anim_speed(2)
7  dv.run(1 / 60.0)
```

Figure 2: Example animation program.

The animation program first creates an object of the class *DesVizMaster* (there should only be a single object of this class), and then calls various methods within this class to set up and run the animation app.

- class *DesVizMaster*(window_width, window_height)
  - Class which contains and provides access to DesViz functionality. There should only be one object instantiated from this class.
  - arguments:
    - **window_width, window_height**: dimensions of the animation window given in pixels (Type: int)
  - methods:
    - *set_paths*(path_fname)
      - Provides a filename *path_fname* (Type: string) for reading the movement paths table if required (see section 4).
    - *set_script*(script_fname)
      - Provides a filename *script_fname* (Type: string) for reading the animation csv file (see section 3).
    - *set_anim_speed*(sim_unit_per_second)
      - Sets the play-back speed of the animation *sim_unit_per_second* (Type: float) if required. If not specified the default play-back speed is 1 unit of simulation time per second. The higher the number the faster the play-back speed.
    - *run*(frame_interval)
      - Starts the animation application and the frame is redrawn every *frame_interval* (Type: float) seconds. In the example shown in Figure 2 the frame will be redrawn every $1/60^{th}$ of a second.

## 3 Animation CSV File Instructions

The animation csv file is structured as shown in Figure 3. The first column gives the simulation time, with the requirement that rows are in chronological order. The second column is for a command/instruction selected from those defined in subsequent subsections. The next seven columns are for data specific to the given instruction. Most instructions do not use all columns for required data.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | time | command | arg1 | arg2 | arg3 | arg4 | arg5 | arg6 | arg7 |
| 2 | 0 | add | background | resources/pmc-backdrop2.png | 1 | 0 | 0 | 1 | |
| 3 | 0 | text_field | plant_label | 0 tons | Arial | 24 | 110 | 275 | |
| 4 | 0 | text_field | sh_label3 | 0 tons | Arial | 24 | 1260 | 30 | |
| 5 | 0 | text_field | sh_label2 | 0 tons | Arial | 24 | 1260 | 275 | |
| 6 | 0 | text_field | sh_label1 | 0 tons | Arial | 24 | 1260 | 500 | |
| 7 | 0 | add_pbar | plant_pbar | 120 | 405 | 100 | 20 | 0 | |
| 8 | 0 | add_pbar | sh_pbar3 | 1302 | 156 | 100 | 20 | 0 | |
| 9 | 0 | add_pbar | sh_pbar2 | 1290 | 402 | 100 | 20 | 0 | |
| 10 | 0 | add_pbar | sh_pbar1 | 1272 | 630 | 100 | 20 | 0 | |
| 11 | 0 | add | truck20-11 | resources/truckF.png | 1 | 0 | 0 | 0 | |
| 12 | 0 | scale | truck20-11 | 0.5 | | | | | |
| 13 | 0 | guide | truck20-11 | 25 | 50 | | | | |
| 14 | 0 | rotation | truck20-11 | 90 | | | | | |
| 15 | 0 | add_pbar | truck20-11pb | 0 | 0 | 50 | 10 | 0 | 0 |
| 16 | 0 | pbar_attach | truck20-11pb | truck20-11 | 65 | 0 | -90 | | |
| 17 | 0 | pbar_color | truck20-11pb | 255 | 0 | 0 | | | |
| ⋮ | | | | | | | | | |
| 20244 | 853.0551775 | image | truck20-31 | resources/truckF.png | | | | | |
| 20245 | 853.0551775 | move_on | truck20-31 | S3toPq | 2.398616453 | 1 | 0 | 1 | |
| 20246 | 853.0551775 | text | sh_label3 | 2800 tons | | | | | |
| 20247 | 853.0551775 | pbar_level | truck20-31pb | 0.539039637 | | | | | |
| 20248 | 853.0551775 | move_on | truck50-33 | S3 | 9.956059344 | 1 | 0 | 1 | |

Figure 3: Example animation csv file shown in Excel.

## 3.1 Animation Settings

### 3.1.1 speed

Sets the speed at which the animation plays in terms of simulation time units per second. The higher the number, the faster the animation.

Arguments:

1. **sim_unit_per_second:** play-back speed (Type: float)

## 3.2 Sprite Properties

This section covers adding and deleting sprites, along with various properties relating to how they appear in the animation.

### 3.2.1 add

This adds an object (i.e. sprite) to be drawn on the animation window, with a specified identifier that can be used to control that sprite in subsequent instructions. In Figure 3, row 2 shows an example of adding a background sprite, and row 11 an example of adding a foreground sprite. Foreground and background are differentiated only by the last argument, 'arg6' in this case. Genreally there should only be one background sprite positioned at (0,0) and would remain stationary for the duration of the animation. This would serve the purpose of a backdrop over which foreground sprites move.

Arguments:

1. **id:** unique identifier for the sprite (Type: string)
2. **image_fname:** filename of the image file of the sprite, must be of a format compatible with pyglet sprites, e.g. png (Type: string)
3. **scale:** scaling factor to resize the sprite with a value of 1 retaining the specified image file dimensions (Type: float)

4. **x0:** (x0,y0) gives the window coordinates (in pixels) of the image origin, i.e. lower-left vertex (Type: float)
5. **y0:** see x0 (Type: float)
6. **is_background:** value of 1 puts sprite in the background layer, and value of 0 puts sprite in the foreground (Type: int 0,1)

#### 3.2.2 delete

This deletes a sprite from the animation canvas. It will no longer be available so any subsequent instructions using that sprite's id will result in an error.

Arguments:

1. **sprite_id:** identifier of the sprite to delete (Type: string)

#### 3.2.3 scale

This sets the scaling factor to resize the sprite with a value of 1 retaining the specified image file dimensions.

Arguments:

1. **sprite_id:** identifier of the sprite to re-scale (Type: string)
2. **scale:** scaling factor (Type: float)

#### 3.2.4 visible

Sets whether the sprite is visible or hidden.

Arguments:

1. **sprite_id:** identifier of the sprite (Type: string)
2. **visible:** value of 1 makes the sprite visible, and value of 0 hides the sprite (Type: int 0,1)

#### 3.2.5 color

This allows the sprite to be drawn with a colour tint. The colour is specified as an RGB tuple of integers '(red, green, blue)'. Each colour component must be in the range 0 (dark) to 255 (saturated).

Arguments:

1. **sprite_id:** identifier of the sprite to colour tint (Type: string)
2. **R:** (R,G,B) is assembled into the tuple of integers which specify the colour tint (Type: int 0..255)
3. **G:** see R (Type: int 0..255)
4. **B:** see R (Type: int 0..255)

#### 3.2.6 image

This allows a sprite's image to be changed during the animation.

Arguments:

1. **sprite_id:** identifier of the sprite (Type: string)
2. **image_fname:** filename of the image file of the sprite, must be of a format compatible with pyglet sprites, e.g. png (Type: string)

### 3.3 Sprite Position and Movement

There are numerous ways in which to influence how a sprite moves in the animation, which are defined in this section.

### 3.3.1 guide

By default, when positioning a sprite on the screen, the sprite's origin (bottom-left corner) is positioned at the specified coordinates. The *guide* instruction allows the simulation programmer to specify something other than the sprite's origin as the reference point to be positioned on the screen. The guide point is specified relative to the sprite's origin at 0 degrees rotation. An example of when this functionality is helpful is when sprites are being moved along a predefined path (see section 3.3.5). In this case, setting the guide point at the sprite centroid will typically result in movement that appears smoother than using the default origin point.

Arguments:

1. **sprite_id:** identifier of the sprite (Type: string)
2. **x:** (x,y) gives the guide point coordinates (in pixels) relative to the sprite origin at 0 degrees rotation (Type: float)
3. **y:** see x (Type: float)

### 3.3.2 rotation

Changes the rotation of the sprite. The sprite will be rotated about its guide point (see section 3.3.1).

Arguments:

1. **sprite_id:** identifier of the sprite (Type: string)
2. **rotation:** clockwise rotation in degrees (Type: float)

### 3.3.3 place

This places a sprite's guide point (see section 3.3.1) at a particular location in the animation window specified by pixel coordinates.

Arguments:

1. **sprite_id:** identifier of the sprite to place (Type: string)
2. **x:** (x,y) gives the window coordinates (in pixels) where to place the sprite's guide point (Type: float)
3. **y:** see x (Type: float)

### 3.3.4 move

This applies a point to point movement to a sprite from its current position to a specified position over a specified time (in simulation time units).

Arguments:

1. **sprite_id:** identifier of the sprite to move (Type: string)
2. **x:** (x,y) gives the window coordinates (in pixels) where to move the sprite's guide point (Type: float)
3. **y:** see x (Type: float)
4. **duration:** time required to complete the movement in simulation time units (Type: float)
5. **auto_orient:** value of 1 will automatically set the rotation of the sprite in the direction of movement, 0 will retain its current rotation (Type: int 0,1)

#### 3.3.5 move_on

This moves a sprite along a predefined path (i.e. sequence of point to point movements, see section4).

Arguments:

1. **sprite_id:** identifier of the sprite to move (Type: string)
2. **path_id:** identifier of the path to move on (Type: string)
3. **duration:** time required to complete the movement in simulation time units (Type: float)
4. **auto_orient:** value of 1 will automatically set the rotation of the sprite in the direction of movement, 0 will retain its current rotation (Type: int 0,1)
5. **start_current:** 1 indicates that the sprite is already part way along the path and will start moving from its current position, 0 will start the movement from the beginning of the path (Type: int 0,1)
6. **end:** proportion of the path to be traversed, value of 1 indicates the path will be traversed to the end (Type: float 0..1)

#### 3.3.6 place_on

This places a sprite at a specified position along a predefined path.

Arguments:

1. **sprite_id:** identifier of the sprite to place (Type: string)
2. **path_id:** identifier of the path to place on (Type: string)
3. **auto_orient:** value of 1 will automatically set the rotation of the sprite in the direction of the path, 0 will retain its current rotation (Type: int 0,1)
4. **end:** position along the path to place the sprite, in terms of the proportion of path length, value of 1 indicates placement at the end of the path (Type: float 0..1)

#### 3.3.7 attach

This attaches one sprite (slave) to another (master), at relative position according to specified offset between the sprite origin points. Both sprites will have the same rotation when being moved together.

Arguments:

1. **slave_id:** identifier of the slave sprite to attach (Type: string)
2. **master_id:** identifier of the master sprite (Type: string)
3. **master_dx:** (master_dx, master_dy) gives the horizontal and vertical offset from the master sprite origin, to the slave sprite origin (Type: float)
4. **master_dy:** see master_dx (Type: float)

#### 3.3.8 detach

This reverses the effect of the *attach* instruction (see section 3.3.7).

Arguments:

1. **slave_id:** identifier of the slave sprite to detach (Type: string)

### 3.4 Text

DesViz also provides instructions for placing text into the animation canvas.

### 3.4.1 text_field

This instruction places a text field at a specified location on the animation window. Various attributes of the text field are supplied such as the font, size and origin coordinates. The default colour of the text label is black.

Arguments:

1. **label_id:** identifier to be associated with the text field (Type: string)
2. **text:** text to be displayed (Type: string)
3. **font_name:** e.g. 'Arial', consult pyglet documentation for recognised fonts (Type: string)
4. **font_size:** size of the font to be used (Type: float)
5. **x:** (x,y) gives the coordinates of the text field's origin (Type: float)
6. **y:** see x (Type: float)

### 3.4.2 text

This allows specific text to be entered into a defined text field (see section 3.4.1).

Arguments:

1. **label_id:** identifier of the text field to be updated (Type: string)
2. **text:** text to be inserted into the text field (Type: string)

### 3.4.3 text_color

This allows the color of a text field to be specified or changed.

Arguments:

1. **label_id:** identifier of the text field to be updated (Type: string)
2. **R:** (R,G,B) is assembled into the tuple of integers which specify the colour of the text displayed (Type: int 0..255)
3. **G:** see R (Type: int 0..255)
4. **B:** see R (Type: int 0..255)

## 3.5 Progress Bars

DesViz provides the ability to display and update progress bars throughout the course of the animation. The progress bars can be statically located on the animation canvas or can be attached to sprites.

### 3.5.1 add_pbar

This adds a progress bar of a given size at a specific position on the animation canvas. The rotation can also be specified. The default colour of the progress bar is blue.

Arguments:

1. **pbar_id:** identifier to be associated with the progress bar (Type: string)
2. **x0:** (x0,y0) gives the coordinates of the progress bar's origin (Type: float)
3. **y0:** see x0 (Type: float)
4. **W:** (W,H) the dimensions at 0 degrees rotation (Type: float)
5. **H:** see W (Type: float)
6. **rotation:** clockwise rotation in degrees (Type: float)

#### 3.5.2 pbar_level

This allows the level of a progress bar to be changed dynamically during the animation.

Arguments:

1. **pbar_id:** identifier of the progress bar to be updated (Type: string)
2. **level:** Value between 0 and 1 specifying how full the progress bar is (Type: float 0..1)

#### 3.5.3 pbar_attach

This allows a progress bar to be attached to move synchronously with a given sprite.

Arguments:

1. **pbar_id:** identifier of the progress bar to be attached (Type: string)
2. **sprite_id:** identifier of the sprite to attach to (Type: float)
3. **x_offset:** (x_offset,y_offset) gives the offset of the progress bar's origin from that of the specified sprite (Type: float)
4. **y_offset:** see x_offset (Type: float)
5. **rotation:** clockwise rotation in degrees relative to the sprite rotation (Type: float)

#### 3.5.4 pbar_color

This allows the colour of a progress bar to be specified.

Arguments:

1. **pbar_id:** identifier of the progress bar to be updated (Type: string)
2. **R:** (R,G,B) is assembled into the tuple of integers which specify the colour of the progress bar displayed (Type: int 0..255)
3. **G:** see R (Type: int 0..255)
4. **B:** see R (Type: int 0..255)

# 4 Defining Movement Paths

As well as basic point to point movements, DesViz also provides functionality to define *paths* which are labelled sequences of point to point movements. The advantage is that a single animation instruction can move a sprite along a path, rather than requiring multiple separate point to point instructions. Paths are defined through a separate input file which can be specified through the *set_paths* method (see section 2.2). In the remainder of this section, the format of the input file for defining paths is specified. Also described is an Excel macro driven spreadsheet to facilitate drawing and defining paths in a point and click manner.

## 4.1 Path Definition File

DesViz can be supplied with a path definition csv file as previously mentioned. The file should have the following columns with headings:

1. **path_id** gives the identifier for the path that the current row relates to (Type: string)
2. **waypt_x** the coordinates (waypt_x, waypt_y) give the next point in the specified path, with the requirement that points are listed in order corresponding to the direction of movement of the path (Type: float)
3. **waypt_y** see waypt_x (Type: float)

Note that paths are strictly directional which is apparent from the structure of the path definition file. Figure 4 shows an excerpt from a path definition csv file shown in Excel. The first path $P$ is essentially just a single point to point movement as it only has 2 way points. The next path *PtoS1q* has a more complicated structure with a sequence of 8 point to point movements.

| | A | B | C |
|---|---|---|---|
| 1 | path_id | waypt_x | waypt_y |
| 2 | P | 285.7368723 | 315.8776287 |
| 3 | P | 134.0083255 | 320.6769587 |
| 4 | PtoS1q | 134.0083217 | 320.6769778 |
| 5 | PtoS1q | 53.1603731 | 351.6543527 |
| 6 | PtoS1q | 86.38542572 | 485.3811919 |
| 7 | PtoS1q | 243.6519376 | 459.8575345 |
| 8 | PtoS1q | 317.8547974 | 351.6543832 |
| 9 | PtoS1q | 541.5712595 | 349.4729404 |
| 10 | PtoS1q | 743.1377512 | 528.793411 |
| 11 | PtoS1q | 949.1344724 | 578.7498975 |
| 12 | PtoS1q | 1251.483962 | 578.7498975 |
| 13 | PtoS2q | 134.0083217 | 320.6769778 |
| 14 | PtoS2q | 53.1603731 | 351.6543527 |
| 15 | PtoS2q | 86.38542572 | 485.3811919 |

Figure 4: Excerpt from a path definition file shown in Excel.

## 4.2 Path Drawing Helper

An Excel macro enabled workbook called *DesVizHelper.xlsm* has been developed to streamline the process of overlaying paths on a backdrop image. It consists of the following worksheets:

1. **ReadMe:** This contains a few pointers, not intended to be comprehensive documentation. This document you are now reading is the best source of information aside from scrutinising the macros.
2. **Control:** There are a few control parameters on this worksheet which are discussed later in this section.
3. **Layout:** This is where the paths are drawn over the top of the background image. Note that paths will be invisible in the DesViz animation.
4. **Segments:** Tabulates data relating to the line segments drawn on the *Layout* worksheet. This table is macro generated and should not require any interaction from the user, other than running the macro to generate the table. It is required for the process of path definition.
5. **Paths:** Tabulates the paths which are incrementally defined by the user. Generally should not require much interaction with the user, however any erroneously entered paths can be manually deleted from this worksheet as required.

The following subsections will run through the process of using DesVizHelper to define paths. All macros required during the process can be accessed through the Menu which can be activated using the short cut CTRL+SHIFT+m, which will open the panel shown in Figure 5.

### 4.2.1 Setting Parameters

Currently there are only 3 parameters to set on the *Control* worksheet, as shown in Figure 6. The first two parameters give the intended dimensions (in pexels) of the animation window (see Figure 2 and section 2.2). These parameters are used to re-scale coordinates from Excel into the animation frame of reference. The benefit of this is that users do not have to be careful about the dimensions of the background image on the *Layout* worksheet, since the coordinates will be re-scaled and flipped (from Excel's top-to-bottom to Pyglet's bottom-to-top) automatically.

The third parameter provides a tolerance in case of line segments being connected imperfectly, i.e., two lines that are intended to be connected do not have adjacent vertices with exactly the same coordinates. This problem is largely avoidable if Excel's snap to shape option is used when drawing line segments. Nevertheless, this parameter is used by a macro titled *JoinSegments* (which can be initiated by clicking the fourth button on the Menu) which checks for any pairs of vertices in close proximity (within the specified tolerance) and then forces one of the vertices to assume the coordinates of the other (the choice of which vertex to modify is somewhat arbitrary within the macro). Another way the tolerance parameter is used is in detecting discontinuities when a path is added, in which case a warning will be printed against the offending point in the *Paths* worksheet.

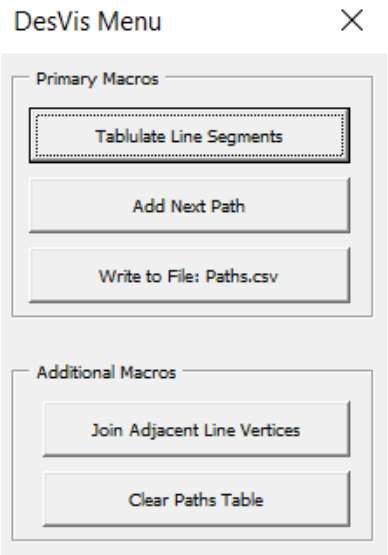


Figure 5: DesVizHelper Menu.

| | A | B | C |
|---|---|---|---|
| 1 | Parameter | Units | Value |
| 2 | animation window width | pixels | 1500 |
| 3 | animation window height | pixels | 750 |
| 4 | line connection tolerance | pixels | 10 |

Figure 6: Control worksheet.

### 4.2.2 Creating the Layout

Creating the layout consists of 2 steps. The first is to paste an image file, which will be the backdrop of the animation, at the origin of the *Layout* worksheet. The second step is to use Excel's standard functionality to draw and connect the lines which will be the backbone of paths. On Excel's ribbon, click *Insert* then *Shapes* and select *Line*.

Figure 7 shows an example, with the background image consisting of some landscape overlaid with tracks (shown in tan colour). The blue lines have then been added over the top of the background image using Excel lines. Note that all the lines are well connected in this example. As mentioned earlier, gaps between adjacent line vertices can be forcibly connected by running the macro *JoinSegments* or by clicking the fourth button on the Menu.

The last step in creating the layout once all lines have been added, is to run the macro *ListSegments* by clicking on the first button on the Menu. This will clear any existing data on the *Segments* worksheet and write a new table giving the vertices of each line on the *Layout*. Note that the coordinates in the table correspond to the animation window coordinate system, not Excel's.

### 4.2.3 Defining Paths

Defining paths consists of 3 steps, however if required the existing data on the *Paths* worksheet may first need to be cleared. This could be done manually, or via the macro *ClearPaths* by clicking on the last button on the Menu. Paths are to be added one at a time, by repeating the 3 steps until all paths have been defined.

Step 1 is to select multiple lines in the correct sequence for the desired path. This can be done by holding down the CTRL button and then clicking the mouse on each line in sequence until the path is fully selected.

Step 2 is to then run the *AddPath* macro, either by clicking the second button on the Menu, or using the shortcut CTRL+SHIFT+p. This will bring up a dialog box requesting a name for the path to be entered. Figure 8 shows a sequence of lines selected (as indicated by highlighted vertices) to be named 'PtoS1q' which has been entered into the dialog box. Either click OK or press Enter to progress to the next step.

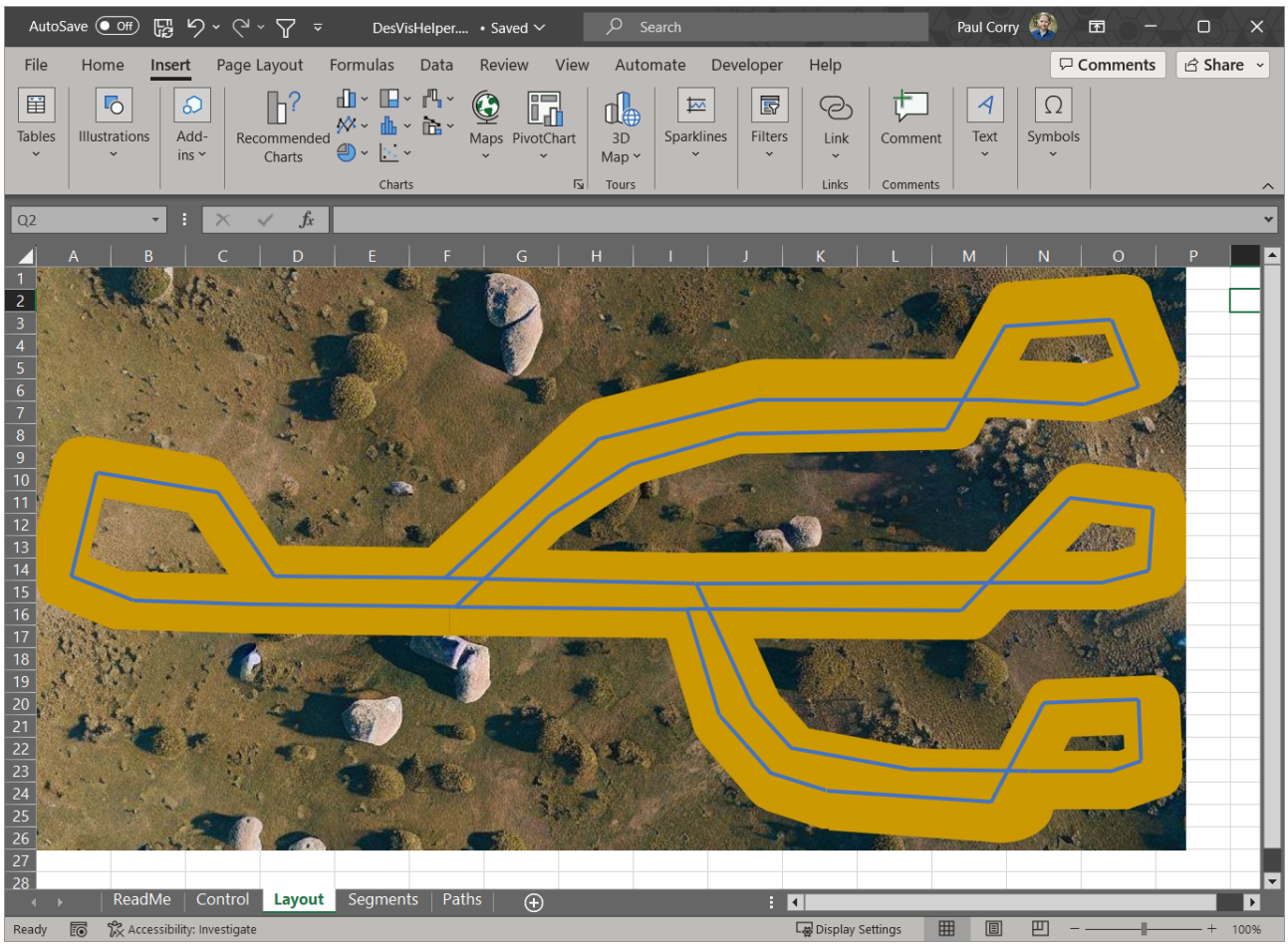


Figure 7: Layout worksheet.

Step 3 is to then specify the direction from which the path starts on the first line segment. The options are L for left, R for right, T for top or B for bottom. In Figure 9, R has been entered because the path starts from the right end of the first line. The user could also have opted to enter B in this case, which would have had the same effect because the line is heading in an upwards direction. Note that this data entry could have been avoided with some extra coding in the macro, but a few edge cases blunted motivation. It may be something that is improved in future.

Once Step 3 is completed by clicking OK or pressing Enter, the new path is added onto the end of the table in the *Paths* worksheet. If there are any gaps between successive lines in the path, due to separated vertices or lines out of sequence, then a warning is written along side the relevant row of data. The relevant line id is also written along side each row, which can be linked to the line id on *Segments* worksheet if any manual cross-checking needs to be done.

### 4.2.4 Writing the Path Definition File

Once all paths have been defined it is straight forward to write the paths table to a file. This can be done with the *WritePathsCSV* macro, or by clicking on the third button in the Menu. The file will be written into the same folder as the DesVizHelper spreadsheet and will be called *Paths.csv*. If there is an existing file of that name it will be overwritten. Note that the 2 extra columns mentioned in the previous section (line id's and warnings) will also be written to the file, however these columns will be ignored by DesViz when the file is read into Python.

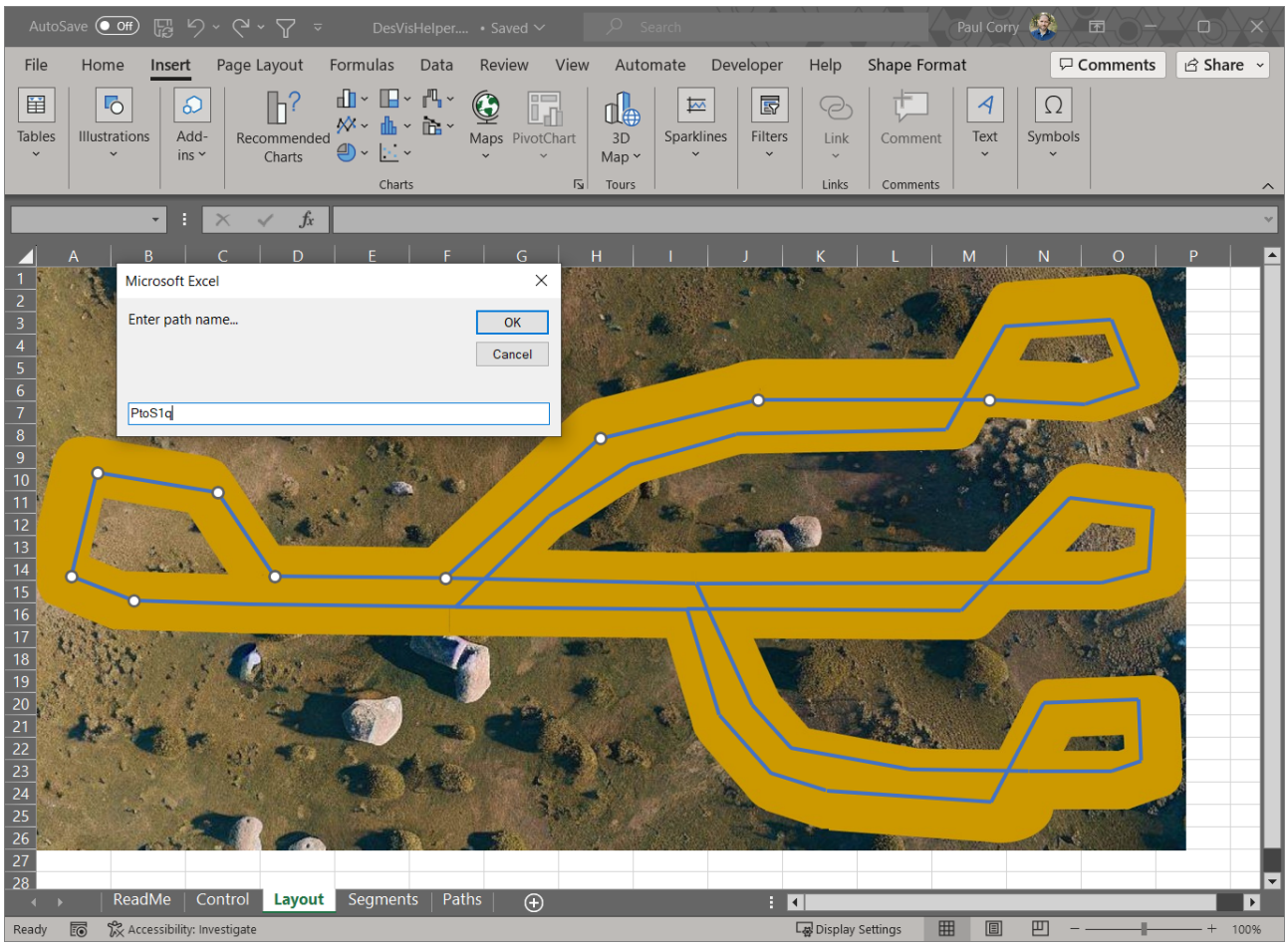


Figure 8: Adding a path steps 1 and 2.

# 5 Future Work

There is much scope to build on this initial version of DesViz. The following are avenues that will be explored for future improvements:

- Secondary window containing animation controls such as play/pause, animation speed, and displaying basic information such as the simulation time.
- Multiple synchronised animation windows allowing for multiple views.
- Zoom and pan functions for animation windows.
- Attachment of text fields to sprites.
- Secondary guide point on sprites allowing for smoother movement between line segments.
- Articulated attachment of sprites allowing for animation of trains or truck trailers.

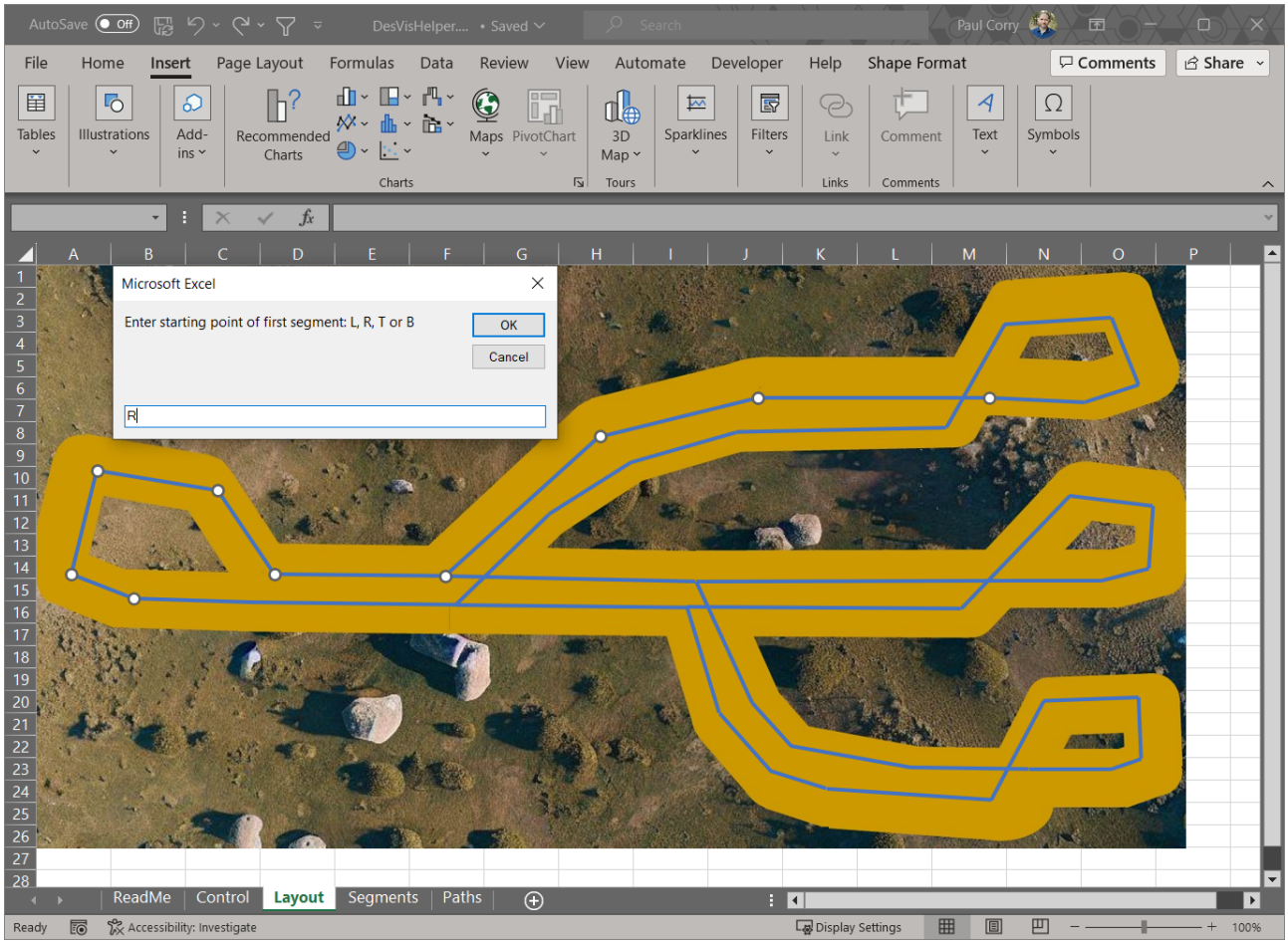


Figure 9: Adding a path step 3.