# Edward Lloyd’s Coffeehouse Project 1 – Pseudocode

## Author: Trevor Hughes – [trevor@trevorhughes.co.uk](mailto:trevor@trevorhughes.co.uk)

## Top Level Description and Objective

This game is intended to be the first in a series of educational games focussing on the subject of risk and insurance.

The game is set in the context of transatlantic shipping in the 1700’s. It was then that the famoush Edward Lloyd’s Coffeehouse became a venue for the ship insurance market.

In this version there is one player operating from one workstation. He is playing against the computer , where the computer is simulating the other player.

The intention of this version is to demonstrate the concept, with a view to:

1. Attracting a financial sponsor
2. Providing a basis for a work scope for a professional game designer.

The commercialised game would be multi-player, working from remote workstations. The core part of the game where ships are crossing the atlantic will continue in real time; player intervention during this period is not essential. Players will be able log back in as many times as they want to in order to view progress, and change their insurance positions if they wish.

## First Level Description of Project 1

There are a number of informative screens which explain the principles of the game. Once understood the player can go directly to “Enter Edward Lloyd’s Coffehouse” to start the game.

The informative screens are:

* Introductory Page – Description of the eductional game series and intent.
* Edward Lloyd’s Coffeehouse – Description of this particular simulation
* Ship Data
* Shipping Routes and Drift – Basis of selection
* Weather and other hazards – Weather patterns and fixed hazards
* Insurance Premiums – How these are negotiated

### Key Programming Concepts and Standard Modules

The programme is currently written in Python 3.12 and requires this to be installed. An attempt will be made to convert this to an executable version so that it can be demonstrated without the player having to set up Python first.

Packages use are :

* pygame
* math
* Random
* numpy
* datetime

The programme also uses a package called Tiled, whip can be found among the standard Python packages. Pycharm found it easily. Tiled works on the conversion of maps to digital input (see below).

### Essential Data

1. **A map of the North Atlantic**. For this trial version a historic map is used. Note that for the commercial version an upgraded map will be required with copyright permission. This is stored in the .venv file)
2. **A list of the ports** used and their co-ordinates on the above map (a pixels on thsize of map displayed). This is stored within the module local\_data.py.
3. **A list of data on all the ships available for selection**. This list includes the ports serviced by the ship, together with data on the ship as extracted from Lloyd’s Register. A key source of information is the Lloyds Register Heritage and Education Centre <https://hec.lrfoundation.org.uk/archive-library/lloyds-register-of-ships-online>.

The data includes age, condition of rigging and hull condition etc. This is stored within the module local\_data.py.

1. **A list of weather events** which will present hazards to the ships. For each event there is a beginning month and an end month to confine the event to an appropriate season, together with starting co-ordinates and trajectory, and wind speed data. Information gleaned from climate enquiries. This is stored within the module local\_data.py.

## Special Modules and Subroutines

**Tiles.py**. Python modules ‘tiles.py’ and ‘spritesheet.py’ enable the conversion of the map and data held in the Tile Map File and converted into json. The Tiled Map File shows a 82 x 60 grid of the North Atlantic with each grid square assigned a sprite. Sea areas are graded according to drift (e.g Gulf Stream). Non-sea areas represent hazards such as rocks and beaches. These are 16 pixel grid squares Note that it is intened to upgrade to 8 pixels grid squares.

**Astar.py.** This is a module copied from code available on the internet for finding the shortest route between two points. In this application only grid squares which represent the sea are available for the route. Further this module has been developed by the author in an appartently unique way. Each grid square is weighted so favour travel according to ocean drift in the general direcation of travel, and impeded by adverse drift. The output of shortest shipping routes broadly resembe those actually used for sailing in the 1700’s.

Blit\_text.py Several modules to blit text in pygame within a text box or ship log.

Append\_if adds data to the log to be blitted only if the preceding lines in the ships log do not contain identical data.

**Text.py** .contains extensive text used by other modules. Storing the text here readily enable modification of the text elements without searching withing the python programmes,

**Subroutines.py** contains all the subroutines, class definitiona and class methods used by programmes elsewhere. Holding these subroutines and classes here facilitates them being called from multiple points within the main programme,

**Local. Data.py** contains all the data on weather events , ships etc used elsewhere in the programme. Storing the text here readily enable modification of the text elements without searching withing the python programmes

Python Modules – General Structure

Before “Running”

1. Imports
2. Pygame.init, window and canvas sizing
3. Color definitions
4. Font Definitions
5. Text and image positions
6. Initial text
7. Variable Initiation
8. Image loading
9. Creation of lists ( which are not dynamically changed after “running”
10. Initiation of objects ( which are not dynamically changed after “running”
11. Rects

After “Running”

1. Load images to ‘canvas’
2. Draw and load recs to ‘canvas’
3. Window blit ‘canvas’
4. Pygame. Event.get() controls such as MouseDown

The ‘canvas’ is first prepared before blitting the ‘canvas’ to the ‘window’

## PSEUDOCODE

## Main.py

|  |  |
| --- | --- |
| Requires list of button names held in local\_data.py  Displays buttons used by the player to display informational screens or start the game.  Defines a class Button. Any button will have x,y coordinates and a width and a height, together with text, a border colour and whether the button has been clicked of not  Defines a method for Buttons which blits the button onto the screen. | Current Implementation |
| Creates constants ‘width’ and ‘height’ for the pygame window | Width=1500, height=1000 |
| Creates constants ‘buttonheight’ and ‘buttonwidth’ | Buttonhieght=50. Buttonwidth=300 |
| Defines coordinates where buttons are to start | ‘buttonstartx=50,’buttonstarty=50 |
| Defines button border colors for the buttons according to the second paramter in the list of buttonsd |  |
| Determine number of button names and blit buttons in a list down the page. | Local\_data.buttonnames[i][0].[1] |
| Inside a ‘running loop’ looks for any of the buttons to be clicked and directs the programme accordingly. |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Files Required |  |
| A .csv representing each cell on the map. The value of each cell represents whether the map at that point is sea, beach, land, or rocks. If sea it also shows the drift |  |
|  |  |
|  |  |
|  |  |
|  |  |

Files required:

1. A Tile Map file is created using Tiler (newmap10spritesupd as Tiled Map File)
2. This is then exported as a .csv file.(as newmap10spritesupd.csv and a spritesheet as a json file (spritesheet.json)

### main.py

‘main.py’ creates an instance of the class Tilemap (see ‘tiles.py’). This requires the name of the .csv and of the spritesheet

## spritesheet.py

is called by tiles.py passing a filename. The filename passed is that of the sprite sheet.

imports pygame and json package

tiles.py

is called from ‘main.py’ and requetes the name of the .csv and the spritesheet..

imports pygame, csv and os

## Premiums\_sub

### First level

In the set up screen the number of ships to be insured is selected.

The number of insurers, in addition to yourself is selected. These insurers are nambed Algo1, Algo2 etc. Each insurer has certain characteristics, there are represented by the algorithms which determine insuers decisions. Each insurer, for this game, has the same maximum amount of insurance cover which is prepared to put at risk. The cost of build of a ship is between £ 2000 and £4000 which would cover a total loss of the ship. The cost of a repair is between £600 and £1400 ples the cost of the cargo. The total amount of insurance coverage available is determined for this game by multiplying the number of ships by £3000, and dividing the insurance available between each of the insurers. Premiums are to cover the ship operations for a full year.

The philosphy of Algo1 is to prefer insuring lower risk ships. Low risk is characterized by ( and in order of):

|  |  |  |
| --- | --- | --- |
| Characteristic | Algo1 Preference | Priority |
| Age | New or recently built | 1 |
| Place of build | 1. UK 2. Europe and America | 4 |
| Hull Condition | A/E | 2 |
| Rig condition | G | 3 |
| Revenus | Largely Indifferent ( mild preference for shorter routes with lower revenues at risk. | 5 |

The philosphy of Algo2 is to prefer insuring ships using shorter routes where (supposedly) they are exposed to less hazards:

|  |  |  |
| --- | --- | --- |
| Characteristic | Algo2 Preference | Priority |
| Age | Indifferent | 5 |
| Place of build | Largely Indifferent | 4 |
| Hull Condition | Largely Indifferent | 3 |
| Rig condition | Largely Indifferent | 2 |
| Shipping Route | Naples, Venice, Stavanger, Venice… | 1 (high to low) |

The phiosophy for the players insurance company (MyAlgo) is for the player to define! An input block will be provided for the player. Meanwhile let’s set MyAlgos strategy to one of insuring longer routes

|  |  |  |
| --- | --- | --- |
| Characteristic | MyAlgo Preference |  |
| Age | Indifferent | 4 |
| Place of build | Indifference | 5 |
| Hull Condition | Indifferent | 3 |
| Rig condition | Largely Indifferent | 2 |
| Shipping Route | Transatlantic and Africa | 1 (low to high) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Minimum | Maximum | Default |
| Number of ships | 5 | 30 | 10 |
| Number of insurers ( in addition to yourself) | 1 | 5 | 2 |
| For default values |  |  |  |
| Maximum insurance to be provided per insurer |  |  | =3000\*10/3  Or 10000 |
| Maximum premium acceptable to ship |  |  | 20% of replacement cost |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

The attitude of all ships is to accept insurance premiums less than 20% of the replacement value. This value is to be adjusted based on experience.

All insurers start by offering premiums of 10% of the RCV of the ship to their most preferred

There are multiple rounds to the auction:

Each round

All 3 insurers list the ships according to their preference ( as above) excluding ships whose insurance has been won.

Where a particular ship is offered a premium by only one underwriter, or where the premium offered is less the ship is assigned that underwriter the insurance for the ship is considered ‘won; The remaining available insurance book for the insurer is reduced, and the ship is no considered in further rounds. Premiums can only if the insurer has the capacity on his book.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable / List Name | Type | Purpose | Comment |
| imax | Integer | Number of ships | Ships to be selected randomly from list in local\_data. Default is 10 |
| kmax | Integer | Number of insurers | Default is 3 |
| Ship\_list\_me | Nested list | List of randomly selected ships with data extracted for local\_data | Number of ships is imax |
| ship\_list\_selected | Nested list | List of instaniated ships derived from ships\_list\_me |  |
| Insurers\_list | Nested list | List of instantiated insurers |  |
| Ship\_list\_nested | Nested list | List of ships with selected data | Unsorted |
| Ship\_sorted\_list | Nested list | List of ships with data sorted |  |
| Placement\_list | list | List of first preferencesm |  |
| Ship\_name\_no\_dup | List | Ships names eliminating duplicates | Must be at least one |
| Diplicates | List | Lists any duplicaates |  |
|  |  |  |  |
|  |  |  |  |