**IN3063 Coursework Task 1**

Develop a simple game consisting of rectangular grid (of size height x width) where each cell has random value between 0 and n.

* **Implementation of the game:**
  + Main menu has been included to allow the selection of the game modes and parameters in a flexible way. When Task 1.py file is executed, Welcome screen will be printed with a list of game types. Normal game type can be chosen for heuristic algorithm and the other for Dijkstra’s.
  + Once the parameters are entered by user, game class will be initialised with the values from the user. This will create a grid which will print the grid.
  + If statement is used to start the game mode according to the user selection.
  + Path and Cost will be printed when the Game mode function finishes executing.

Shape, rectangle

Description automatically generated

Figure: Menu

* **Develop Heuristic Algorithm:**
  + 2 functions have been developed that includes heuristic algorithm to find the shortest path (one for each game mode). The function will start with a while loop that will run until the last cell in the grid is reached. 4 variables are getting initialised at the start of the while loop. 2 of them is a Boolean to keep track if movement toward down or right is available. Other 2 variables are to keep track of the cost of neighbours.
  + If statement is used to check if cell is available to move. If the statement returns true, It will assign the cost of that cell and assign True which indicates cell availability.
  + Another if statement is used to check if cells are available in both directions. If true, it will compare the cost of both cells and move to the one with less. If only one cell is available, it will move towards that direction.
  + When moved to a cell, position will be updated as well as the cost.
  + Once the while loop ends, the function will return the path and total cost.
* **Implement Dijkstra’s algorithm:**
  + Dijkstra’s algorithm is an algorithm that finds the shortest path in between points in an efficient way.
  + There are many ways to implement this algorithm, I have chosen to implement using priority queues.
  + A function is created for this algorithm. This function starts by calling another function called find\_neighbours. Find\_neighbours finds all the neighbours with its costs and returns it in a dict object. Once this function is assigned to an object inside Dijkstra function, it will start a for loop which will create a dict object with all the values of the grid as keys set to a large number. This is to keep track of unvisited cells.
  + It will then use a while loop until unvisited dict object is empty.
  + Inside while loop minNode object is getting assigned to the smallest in unvisited which is found by usining min function.
  + With a for loop, all the neighbours are visited and set to a tempDist object.