2801ICT Assignment 2

Damon Murdoch s2970548

[1. Problem Statement 3](#_Toc482211824)

[2. User Requirements 3](#_Toc482211825)

[3. Software Requirements 3](#_Toc482211826)

[4. Software Design 4](#_Toc482211827)

[4.1 High Level Design 4](#_Toc482211828)

[4.1.1 Laddergram 4](#_Toc482211829)

[4.1.2 Rectangles 5](#_Toc482211830)

[4.2 UML Diagram 6](#_Toc482211831)

[4.2.1: Laddergram UML Diagram 6](#_Toc482211832)

[4.2.2: Rectangles UML Diagram 6](#_Toc482211833)

[4.3 Functions List 6](#_Toc482211834)

[4.3.1 Laddergram 6](#_Toc482211835)

[4.3.2 Rectangles 7](#_Toc482211836)

[4.4 Data Structures 8](#_Toc482211837)

[4.4.1 Laddergram 8](#_Toc482211838)

[4.4.2 Rectangles 9](#_Toc482211839)

[4.5 Detailed Design 9](#_Toc482211840)

[4.5.1 laddergram.py 9](#_Toc482211841)

[4.5.1.1 Get Heuristic Function 9](#_Toc482211842)

[4.5.2.2 Solve Function 10](#_Toc482211843)

[4.5.2 Rectangles.py 11](#_Toc482211844)

[4.4.2.1 Check Intersect 11](#_Toc482211845)

[4.4.2.2 Check Contains 11](#_Toc482211846)

[37. Requirement Acceptance Tests 11](#_Toc482211847)

[5.1 Laddergram.py 11](#_Toc482211848)

[5.2 Rectangles.py 12](#_Toc482211849)

[38. Detailed Software Testing 13](#_Toc482211850)

[6.1 Laddergram.py 13](#_Toc482211851)

[6.2 Rectangles.py 14](#_Toc482211852)

[39. User Instructions 15](#_Toc482211853)

[7.1 Laddergram.py 15](#_Toc482211854)

[7.2 Rectangles.py 16](#_Toc482211855)

# Problem Statement

The goal of this project is to write two programs, one for solving ladder-grams and the other for identifying intersecting rectangles. The first application is an implementation of a ‘Ladder-Gram’ Solver, which involves the user entering a starting word and a target word. If possible, the starting word must be transformed into the target word through changing a single character at a time. The second program is that given two rectangles of any size or rotation, find if they intersect or if one is inside the other. In addition to this, the program should also be able to find the area of the intersection between rectangles if one exists.

# User Requirements

The following outlines the user requirements of the program.

**Laddergram:**

* Input starting and target word through command line
* Any character cases can be used in inputs
* Rectangles can be given any rotation

**Rectangles:**

* Enter two rectangles using point values in the command line
* Randomly generate rectangles
* Save and interact with resulting graph

# Software Requirements

**Laddergram:**

* Transform source word to target word
* All words must be in the dictionary file
* Only change one letter in a single step
* Print full path taken to reach target word

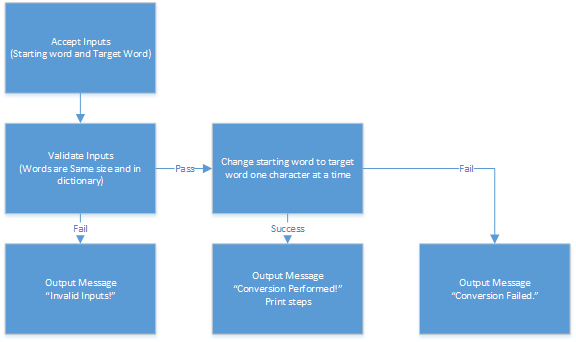
**Rectangles:**

* Random rectangle generation
* Detect overlapping rectangles
* Detect number of overlaps
* Detect one rectangle contained within the other
* Display rectangles and results in a graph
* Provide the area of intersections for the two triangles.

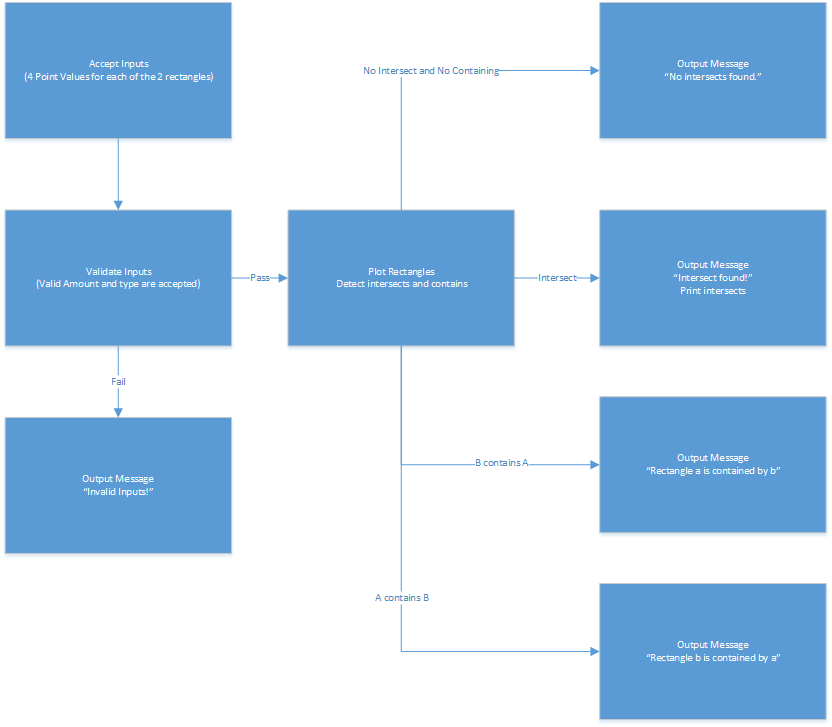
# Software Design

## 4.1 High Level Design

### 4.1.1 Laddergram

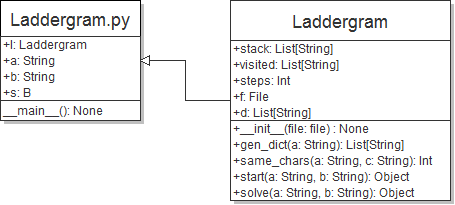


### 4.1.2 Rectangles

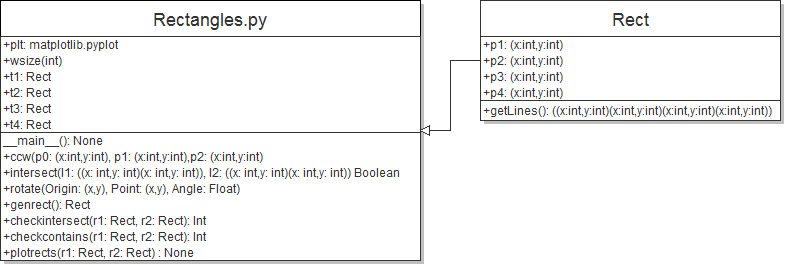


## 4.2 UML Diagram

### 4.2.1: Laddergram UML Diagram



### 4.2.2: Rectangles UML Diagram



## 4.3 Functions List

### 4.3.1 Laddergram

**Main Program**

\_\_main\_\_():

Description: Code executed outside of a function in Laddergram.py  
Input: None.

Side-Effects: Initialises and reads strings a and b, as well as the Laddergram l. Runs start(a,b) and prints a string depending on if solve was successful.  
Output: No return value.

**Laddergram Class:**

+\_\_init\_\_(file):

Description: Initialises the laddergram object with a dictionary filename to read words from.  
Input: String filename.

Side-Effects: Initialises self.stack([String]), self.visited ([String]) , self.steps (int), and self.f (File)  
Output: No return value.

+gen\_dict(a):

Description: Generates a list of all the words in ‘f’ that are the same length as a, and returns it.   
Input: String a.

Side-Effects: None.  
Output: List storing all words with the same length as a.

+get\_heuristic(a,b):

Description: Returns the number of characters a shares with b.  
Input: String a, String b

Side-Effects: None.  
Output: Returns an integer with a starting value of 0, which is incremented for each index in a and b that have the same value.

+start(a,b):

Description: Initialises or resets relevant values to the solve function, and then calls it.  
Input: Strings a,b.

Side-Effects: calls generate dictionary, Resets steps to 0, empties stack and visited, calls solve(a,b).  
Output: Output of solve(a,b).

+solve(a,b):

Description: Recursively backtracks using depth-first search to find a pathway from a to b, by changing a single character at a time to other words in the dictionary.   
Input: Strings a,b.

Side-Effects: Appends to self.stack, self.visited.  
Output: -1 if function fails, self.stack if function succeeds.

### 4.3.2 Rectangles

+ccw(p0,p1,p2):

Description:

Input:  
Side-Effects:

Output:

+intersect(l1,l2):

Description: Calculates if two given lines intersect one another.

Input: l1, l2: Line values stored as a tuple. ((x,y),(x,y))  
Side-Effects: None.

Output: Returns a boolean True if the lines intersect, and boolean false if they do not.

+rotate(origin,point,angle):

Description: Rotates a point by a given angle with reference to a given origin point.

Input: Tuple Origin (x,y) Tuple Point(x,y) Float Angle  
Side-Effects: None

Output: int qx, qy: New x and y positions for given point

+genrect():

Description: Generates a random rectangle for testing.

Input: None.  
Side-Effects:None.

Output: Rectangle with randomised points.

+checkintersect(r1,r2):

Description: Utilises the intersect function to check if two rectangles intersect at any point.

Input: Rectangles r1, r2.   
Side-Effects:None.

Output: Returns true if an intersect is found, and false if no intersect is found.

+checkcontains(r1,r2):

Description: Utilises the intersect function to check if r1 is located inside r2.

Input: Rectangles r1, r2.  
Side-Effects: None.

Output: Returns true if r1 is contained in r2, and false if it is not.

+plotrects(r1,r2);

Description: Plots the given rectangles on a python matplot, and displays an evaluation of the rectangle interaction as the title.

Input: Rectangles r1, r2.  
Side-Effects: None.

Output: None.

## 4.4 Data Structures

### 4.4.1 Laddergram

**Main Program:**

None.

**Laddergram Class:**

Stack:

Structure Type: List

Description: List storing all of the steps taken to reach the goal state.

Members: String: Values of each state of ‘a’ as it is converted to ‘b’.

Functions: Solve, Start, \_\_Main\_\_

Visited:

Structure Type: List

Description: List storing all of the values which have been visited by the solve function, and prevents them from being re-visited.

Members: String: Word which has been visited.

Functions: Solve

D:

Structure Type: List

Description: List storing every word read from dictionary.txt that is of the same length as the input words a and b.

Members: String: Words from dictionary with same length as a and b.

Functions: gen\_dict, solve

### 4.4.2 Rectangles

No abstract data structures were used in this program.

### 4.5 Detailed Design

#### 4.5.1 laddergram.py

##### 4.5.1.1 Get Heuristic Function

get\_heuristic(String a, String b)

h = 0

If a is not the same length as b:

Return -1

Loop for every char in a:

If char at a[i] is equal to char at b[i]:

h++

Return h

#### 4.5.2.2 Solve Function

solve(String a, String b)

out = a

Self.steps += 1

Lock = new List

Add = True

Loop for i in range(Length of a):

If a and b have the same character at position ‘i’, lock that character

If a and b have differing lengths or either a or b is not in the dictionary:

print(“Invalid inputs!”)

Return -1

If a = b:

Return self.stack

S = new List

For i in length of dictionary:

If index ‘i’ in dictionary has not been visited and only differs by ‘a’ by one character:

For j in range length of a

If locked letter is different in word:

Add = False

If Add = True:

Generate heuristic of dictionary[i]

Add tuple (dictionary[i], heuristic) to s

Add = True

Sort ‘s’ by highest heuristic value first

If there are more than zero values in ‘s’:

Out = s[0] # Highest Heuristic Value

Append out to stack

Append out to visited

Solve (out, b)

Else if there greater than zero values in the stack:

Solve(stack.pop(),b)

Else:

Return -1 #Unsolvable position

#### 4.5.2 Rectangles.py

##### 4.4.2.1 Check Intersect

checkintersect(Rectangle r1, Rectangle r2)

Interects = 0

S1 = all lines in r1

S2 = all lines in r2

For i in range s1:

For j in range s2:

If intersect at s1[i], s1[j] then

Intersect+=1

Return intersects

##### 4.4.2.2 Check Contains

checkcontains(Rectangle r1, Rectangle r2)

Intersects=0

s=r1.get\_lines()

Ln = Line(Arbitrary point in r1, same point with y value + 1000)

Loop for i in range length(s):

If intersect(ln,index i of s]):

Intersects++

Return intersects # If this function returns one and only one, r1 is contained within r2.

# Requirement Acceptance Tests

### 5.1 Laddergram.py

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement Number** | **Test** | **Implementation** | **Result** | **Comments** |
| 1 | Input starting and target word through command line | Full | Pass | - |
| 2 | Using any character cases in inputs | Full | Pass | - |
| 3 | Source word successfully transforms to target word where it is possible | Full | Pass | - |
| 4 | ‘Lead’ to ‘Gold  In 3 steps | Full | Pass | - |
| 5 | ‘Hide’ to ‘Seek’ in 6 steps | Full | Pass | - |
| 6 | All entered words must be in the dictionary file | Full | Pass | - |
| 7 | Only singular letters changed in each step | Full | Pass | - |
| 8 | Print full path taken to reach target word | Full | Pass | - |

### 5.2 Rectangles.py

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement Number** | **Test** | **Implementation** | **Result** | **Comments** |
| 1 | Enter two rectangles, providing point values at command line | Full | Pass | - |
| 2 | Randomly Generate two rectangles | Full | Pass | - |
| 3 | Rectangles can be given any rotation | Full | Pass | - |
| 4 | Successfully detect no overlap | Full | Pass | - |
| 5 | Successfully detect overlapping rectangles | Full | Pass | - |
| 6 | Detect number of times overlap occurs | Full | Pass | - |
| 7 | Detect if one rectangle is stored within another | Full | Pass | - |
| 8 | Display rectangles and results visually in a graph | Full | pass | - |
| 9 | Provide the area of intersections for the two triangles. | None | Fail | Not Implemented |

# Detailed Software Testing

### 6.1 Laddergram.py

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Test** | **Expected Results** | **Actual Results** |
| **Inputs** | | | |
| 1 | Input ‘123’ | Print “Invalid Inputs! Solution not reached.” | Expected Results |
| 2 | Input ‘ ‘ | Print “Invalid Inputs! Solution not reached.” | Expected Results |
| 3 | Input ‘asdfghjkl’’ | Print “Invalid Inputs! Solution not reached.” | Expected Results |
| 4 | Input ‘Lead’ and ‘Gold’ | Solution found! Load Goad Gold Changes: 3 | Expected Results |
| 5 | Input ‘Hide’ and ‘Seek’ | Solution found!  Side  …  Seek  Changes: 6 | Solution found! Side Sade Sabe Sabs Secs Sees Seek Changes: 7 |
| 6 | Input ‘Hello’ and ‘Goodbye’ | Print “Invalid Inputs! Solution not reached.” | Expected Results |
| 7 | ‘Hello’ and ‘asdfg’ | Print “Invalid Inputs! Solution not reached.” | Expected Results |

### 6.2 Rectangles.py

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Test** | **Expected Results** | **Actual Results** |
| **Inputs** | | | |
| 1 | Rectangles with values:  **R1:**  (4,4),(16,4),  (4,16),(16,16)) **R2:**  (8,8),(12,8),  (8,12),(12,12)) | Print “Rectangle is contained within other rectangle.” | Expected Results |
| 2 | Rectangles with values:  **R1:**  (14.5, 1.6)(8.5, 5.0)  (14.6, 9.5) (8.5, 12.9)  **R2:**  (16.1, 2.3) (8.5, 4.7)  (16.1,10.8) (8.5,13.3) | Print “Intersect found! Intersects: 5 | Expected Results |
| 3 | Rectangles with values:  **R1:**  (11,1),(7,4.5),  (12,6),(7,9.5)  **R2**  (19.5,12),(10,9),  (19.5,17.8),(10,15) | Print “No intersections found.” | Expected Results |
| 4 | Rectangles with values:  **R1:**  (3.9,3.7),(7.5,9.7),(3.9,0.2),(7.5,6.2)  **R2:**  (10.4,10.1),(5.0,7.4),  (10.3,17.4),(5.0,14.5) | Print “Intersect found! Intersects” 2 | Expected Results |

# User Instructions

### 7.1 Laddergram.py

1. Extract the archive
2. Ensure the file ‘dictionary.txt’ is stored inside the same directory as Laddergram.py.
3. Execute the application ‘Laddergram.py’.
4. Enter a starting word in the command window
5. Enter a finishing word in the command window
6. If a solution can be found, it will be displayed in the command window. Otherwise, the text “Solution not reached.” will display.
7. Press any key to exit the program.

### 7.2 Rectangles.py

1. Extract the Archive
2. Execute application ‘Laddergram.py’
3. Two randomly generated rectangles will appear in a window, as well as the dimensions in the command line.
4. Close the window, and the next graph will appear
5. The screen will now display a graph with one cube inside another, to demonstrate that the contains function is functional.
6. Close this window to close the application.