COMP1001 – Project Report

Name: SHOAIB Muhammad **Student ID:** 18079999D
Program Name: Broad Discipline of Computing

Problem Description:
We have 3 couples with different coloured clothes to distinguish each couple on the east side of the river. The couples need to cross the river (go from east to west), but there are some conditions for them to cross:

- ✓ A boat can only accommodate a maximum of **2 people** at a time.
- ✓ A third person trying to enter will **break** the boat, and no one can swim to cross the river.
- ✓ A husband cannot let his wife **be with another man** in his own **absence**.
- ✓ However, wives can stay on either side in absence of their husbands **only when** there are no other men on that side of the river.
- ✓ The state is also **illegal** if the **boat is not on the same side as the people** (if all of them are at one side and the boat is on the other side) since we are including boat's state in all of the states.

We need to write a python program to find shortest possible path for the three couple to cross without violating any of the conditions given above.
Before going towards the implementation, let us perform the abstraction to ease the implementation part.

Data Abstraction:

Key
gw: green wife
gh: green husband
rw: red wife
rh: red husband
bw: blue wife
bh: blue husband

- ✓ Location of each entity is either 'E' or 'W'.
- ✓ State of the system: ('E/W', 'E/W', 'E/W', 'E/W', 'E/W', 'E/W', 'E/W')

$$\begin{tikzcd}[column sep=small] \text{'E/W'} & \text{'E/W'} & \text{'E/W'} & \text{'E/W'} & \text{'E/W'} & \text{'E/W'} & \text{'E/W'} \\ \text{gw} \arrow[u] & \text{gh} \arrow[u] & \text{rw} \arrow[u] & \text{rh} \arrow[u] & \text{bw} \arrow[u] & \text{bh} \arrow[u] & \text{boat} \arrow[u] \end{tikzcd}$$

- ✓ There are altogether 128 states (2^7 = 128).
- ✓ Due to the problem constraints, there are only 42 legal states.

Data Needed:

1. **Boolean data** for states since two states only, 'E' or 'W'.
2. **A set of 7 Boolean data** to represent the states of all entities defined above.
3. A **graph** consisting of a set of nodes and a set of corresponding links.
4. A **sequence** for the shortest path (we need a sequence because we need to keep the state transitions/changes in an order).

Algorithm needed to solve the problem and its modular design:

```
function solver( )
    input: none
    output: none (print a human readable version of the path)
    S ← genStates( )
    G ← genGraph(S)
    start = "EEEEEEE"
    dest = "WWWWWWW"
    P ← genShortestPath(G)
    genTrip(P)
```

This function will solve the problem.

Function signatures:

```
function genStates( )
    input: none
    output: return a set of all possible states
```

```
function genShortestPath(G,s,d)
    input: G:graph, s:starting point, d:destination
    output: shortest path from source 's' to
            destination 'd' of minimum distance
```

```
function genGraph(S)
    input: S, a set of states
    output: return a graph connecting all the
            legal states in S
```

```
function genTrip()
    input: shortest path for solution to problem
    output: none (print the path in a human
            readable format)
```

The genGraph function is still too large to be a single function.

```
function genGraph(S)
    for each n in S do
        if n is legal do
            add n to set of legal nodes (L)
    for each m in L so
        find the set of L's neighbouring nodes (L(m))
        add L(m) to graph G
    return G
```

```
function isLegal(S)
    input: S: a state
    output: return True if legal, False if illegal
```

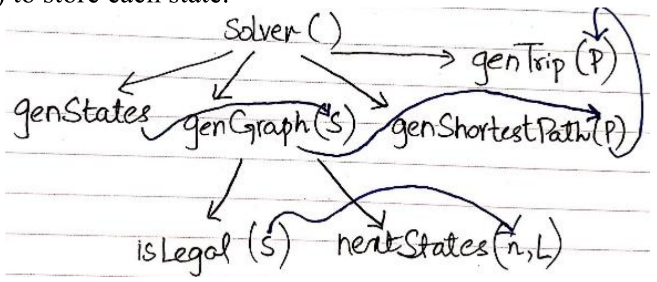
```
function findNextStates(n,L)
    input: n: a state, L: a set of legal states
    output: return a set of n's neighbouring
            states in L
```

<u>Python Implementation of Data Types:</u>

1. States ("E", "W") as Boolean values → could be Boolean values in python
2. A set of seven Boolean values → could be a tuple of strings (because once set, we do not need to change them, so it is better to use tuple since it is immutable and will avoid accidental operations or alterations to the set of the Boolean values).
3. A graph → could be implemented in python using dictionary data structure (we could use a list of lists but since it is not effective, we use a dictionary to simplify our program).
4. A sequence of (s), the shortest path → can be implemented by using a list (cannot use tuple since we need to append the states to the list).

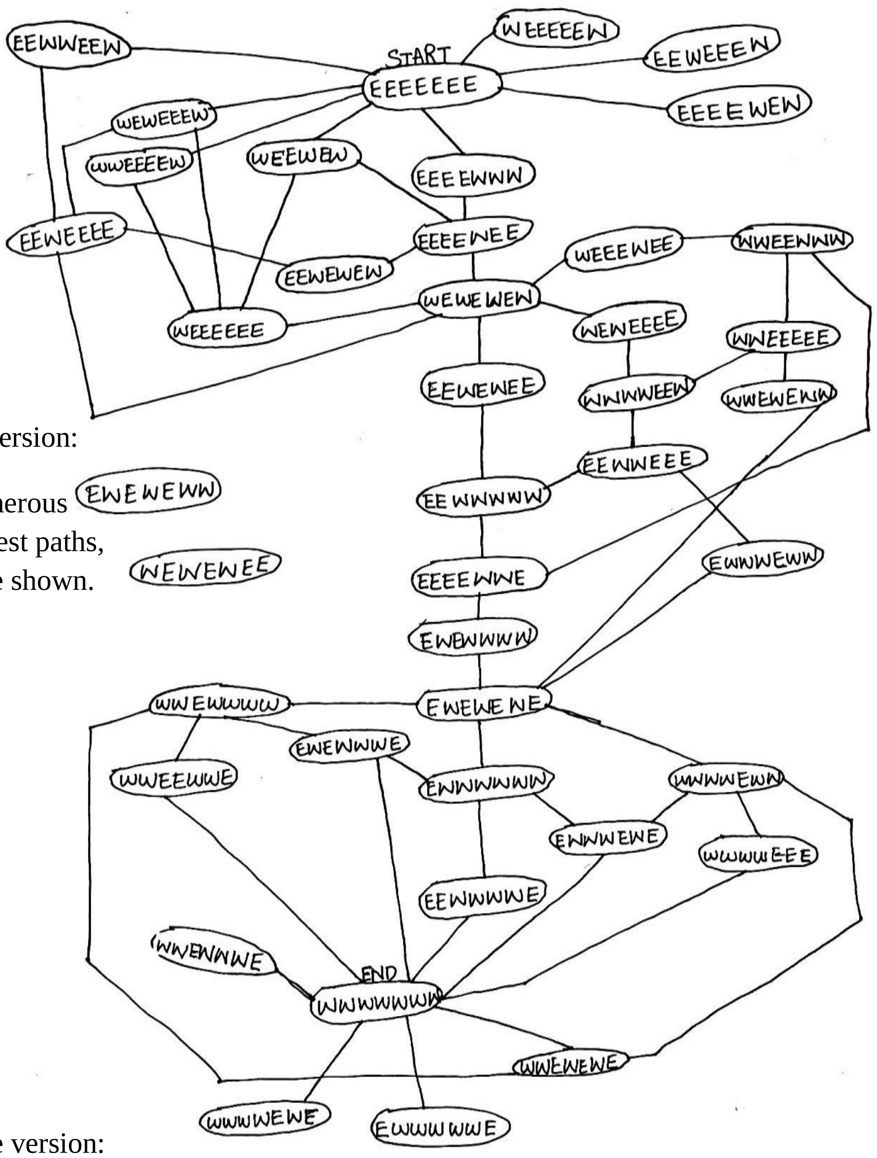
Overview:

- ✓ To solve the problem, we use a modular approach since the algorithm is not that simple.
- ✓ We use 7 Boolean values ("E"/"W") to store each state.
- ✓ The modular design summarises to:



- ✓ After dividing the algorithm into modules and having a rough overview of how we must implement them in python, we move towards implementing the algorithm using python.

Graph:



Abstraction Version:

There are numerous possible shortest paths, all of them are shown.

More readable version:

```
EEEEEEE  :  {'WWEEEEW', 'WEWEEEW', 'EEWEEEW', 'EEEEWEW', 'EEWWEEW', 'WEEEWEW', 'WEEEEEW', 'EEWEWEW', 'EEEEWWW'}
EEEEWEE  :  {'WEWEWEW', 'WEEEWEW', 'EEWEWEW', 'EEEEWWW'}
EEEEWEW  :  {'EEEEEEE'}
EEEEWWE  :  {'EWEWWWW', 'EEWWWWW', 'WWEEWWW'}
EEEEWWW  :  {'EEEEWEE', 'EEEEEEE'}
EEWEEEE  :  {'EEWWEEW', 'WEWEWEW', 'WEWEEEW', 'EEWEWEW'}
EEWEEEW  :  {'EEEEEEE'}
EEWEWEE  :  {'WEWEWEW', 'EEWWWWW'}
EEWEWEW  :  {'EEWEEEE', 'EEEEWEE', 'EEEEEEE'}
EEWWEEE  :  {'WWWWEEW', 'EWWWEWW', 'EEWWWWW'}
EEWWEEW  :  {'EEWEEEE', 'EEEEEEE'}
EEWWWWE  :  {'EWWWWWW', 'WWWWWWW'}
EEWWWWW  :  {'EEWWEEE', 'EEEEWWE', 'EEWEWEE'}
EWEWEWE  :  {'EWEWWWW', 'EWWWWWW', 'WWWWEWW', 'WWEWEWW', 'EWWWEWW', 'WWEWWWW'}
EWEWEWW  :  {}
EWEWWWE  :  {'WWEWWWW', 'EWWWWWW', 'WWWWWWW'}
EWEWWWW  :  {'EEEEWWE', 'EWEWEWE'}
EWWWEWE  :  {'EWWWWWW', 'WWWWWWW', 'WWWWEWW'}
EWWWEWW  :  {'EEWWEEE', 'EWEWEWE'}
EWWWWWE  :  {'WWWWWWW'}
EWWWWWW  :  {'EEWWWWE', 'EWEWWWE', 'EWWWEWE', 'EWEWEWE'}
WEEEEEE  :  {'WWEEEEW', 'WEWEWEW', 'WEEEWEW', 'WEWEEEW'}
WEEEEEW  :  {'EEEEEEE'}
WEEEWEE  :  {'WEWEWEW', 'WWEEWWW'}
WEEEWEW  :  {'WEEEEEE', 'EEEEWEE', 'EEEEEEE'}
WEWEEEE  :  {'WEWEWEW', 'WWWWEEW'}
WEWEEEW  :  {'WEEEEEE', 'EEWEEEE', 'EEEEEEE'}
WEWEWEE  :  {}
WEWEWEW  :  {'EEEEWEE', 'WEWEEEE', 'WEEEWEE', 'EEWEEEE', 'WEEEEEE', 'EEWEWEE'}
WWEEEEE  :  {'WWEWEWW', 'WWWWEEW', 'WWEEWWW'}
WWEEEEW  :  {'WEEEEEE', 'EEEEEEE'}
WWEEWWE  :  {'WWEWWWW', 'WWWWWWW'}
WWEEWWW  :  {'WWEEEEE', 'EEEEWWE', 'WEEEWEE'}
WWEWEWE  :  {'WWEWWWW', 'WWWWWWW', 'WWWWEWW'}
WWEWEWW  :  {'WWEEEEE', 'EWEWEWE'}
WWEWWWE  :  {'WWWWWWW'}
WWEWWWW  :  {'EWEWWWE', 'WWEWEWE', 'WWEEWWE', 'EWEWEWE'}
WWWWEEE  :  {'WWWWWWW', 'WWWWEWW'}
WWWWEEW  :  {'EEWWEEE', 'WWEEEEE', 'WEWEEEE'}
WWWWEWE  :  {'WWWWWWW'}
WWWWEWW  :  {'WWEWEWE', 'WWWWEEE', 'EWWWEWE', 'EWEWEWE'}
WWWWWWW  :  {'WWEWWWE', 'EWEWWWE', 'WWEWEWE', 'EWWWWWE', 'EEWWWWE', 'WWWWEWE', 'WWWWEEE', 'EWWWEWE', 'WWEEWWE'}
```