Name:

Student ID:

10.009 The Digital World

Term 3. 2016.

Final Exam

April 28, 2016 (Thursday) 3:00-5:30pm. Room: Cohort Classroom

Most recent update: April 24, 2016

- Write your name and student ID at the top of this page.
- This exam has three parts. First try to complete Part A then move on to Part B and Part C. You need not do the question in sequence. Try to do all the questions. You can secure a maximum of 100 points if you complete Part A to C. Part D is optional and you can get additional 15 bonus points.
- For Part A (questions 1-2), write your answers on this exam paper.
- For Part B to D (question 3-9), submit your solutions to Tutor.
- You may consult all material on your laptop and in your notes. You may also use any book(s) as a reference.
- **You are not allowed to use any Internet accessing or communicating device during the exam.**
- **You are not allowed to consult anyone inside or outside of the classroom other than the proctors in the examination room.**
- Use IDLE / Canopy to test your programs. Once you are satisfied, enter your program into Tutor and either save it or submit. In case you decide to save, then you MUST submit it before the end of the exam.
- All answers will be graded manually. You may be able to earn partial credit for questions.
- Good luck!

Summary:

Category	Description	Number of Problems	Total Points
Part A	Written questions	2 (Questions 1-2)	25
Part B	Programming question	2 (Questions 3-4)	25
Part C	Longer Programming questions	3 (Questions 5-7)	50
Part D	Optional (Bonus point) questions	2 (Question 8-9)	15

Q1	
Q2	
SubTotal	

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN
HONOUR CODE

"As a member of the SUTD community, I pledge to always uphold honourable conduct. I will be accountable for my words and actions, and be respectful to those around me."

Introduction to the SUTD Honour Code
What is the SUTD Honour Code?
The SUTD Honour Code was established in conjunction with the school's values and beliefs, in good faith that students are able to discern right from wrong, and to uphold honourable conduct. It is an agreement of trust between the students and the staff and faculty of SUTD, and serves as a moral compass for students to align themselves to. Being in a university that aspires to nurture the leaders of tomorrow, it calls for students to behave honourably, not just solely in their academic endeavours, but also in everyday life.

What the Honour Code encompasses
Integrity & Accountability
To be honourable is to do what is right even when nobody is watching, and to be accountable for the things one does. One should always be accountable for one's words and actions, ensuring that they do not cause harm to others. Putting oneself in a favourable position at the expense of others is a compromise of integrity. We seek to create a community whereby we succeed together, and not at the expense of one another.

Respect
Part of being honourable is also respecting the beliefs and feelings of others, and to never demean or insult them. Should conflicts arise, the aim should always be to settle them in a manner that is non-confrontational, and try to reach a compromise. We will meet people of differing beliefs, backgrounds, opinions, and working styles. Understand that nobody is perfect, learn to accept others for who they are, and learn to appreciate diversity.

Community Responsibility
In addition to that, being honourable also involves showing care and concern for the community. Every individual has a duty to uphold honourable conduct, and to ensure that others in the community do likewise. The actions of others that display immoral or unethical conduct should not be condoned nor ignored. We should encourage each other to behave honourably, so as to build a community where we can trust one another to do what is right.

Student's signature

Part A

Q.1 [15 points]

This question is related to graphical user interface (GUI) programming using Kivy.

a) Explain what App.build() method does and state what this method should return. [5 pts]
b) Explain what widget.bind() method does and its argument. [5pts]
c) What are the classes needed to create an application with several Screens? Show their relationship with each other. [5 pts]

Your Answer:

Q.2 [10 points]

In Week 9, you worked on a boundary follower problem in your 1D Mini Project.

a) State the minimum number of sonar sensors needed in this problem. Which are the minimum sonar sensors needed and explain why? [5 pts]
b) The minimum number of states in this problem is two. State the two states and explain why at least two states are needed. [5 pts]

Your Answer:

Part B

Q.3 [10 points]

Write function that takes in a non-empty string that contains numbers and returns a list of the numbers with the maximum occurrences and the maximum count of these numbers. If there is more than one number having the same maximum count, the function should return the list in an ascending order. Note that the numbers in the output list are of **int** type. Hint: You can use **sorted()** to sort the list.

Sample Tests:

```
print 'test 1'
inp='2 3 40 3 5 4 -3 3 3 2 0'
print maxOccurrences(inp)

print 'test 2'
inp='9 30 3 9 3 2 4'
print maxOccurrences(inp)
```

Output:

```
test 1
([3], 4)

test 2
([3, 9], 2)
```

Q. 4 [15 points]

Create a state machine class called **RingCounter** using **sm.SM** that simulates a 3-bit ring counter. The ring counter has the following description:

- The input is a reset signal which can either be 0 or 1.
- The output is a 3-bit binary string of the ring counter.
- When the reset signal is 1, the output is '000'.
- When the reset signal is 0, the output increases by 1 in binary form. It cycles from '000' (0) to '001' (1), '010' (2), '011' (3), '100' (4), '101' (5), '110' (6), '111' (7), and goes back to '000' (0).
- Notice that when the output reaches the maximum value '111' (7), it goes back to '000' (0).

The binary representation of a decimal number is given here:

Decimal	Binary	Decimal	Binary
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Sample Tests:

```
print 'test 1'
r=RingCounter()
print r.transduce([0,0,0,0,0,0,0,0,0])

print 'test 2'
r=RingCounter()
print r.transduce([0,0,0,1,0,0,0,0,0])

print 'test 3'
r=RingCounter()
print r.transduce([0,0,0,1,0,0,1,0,0])
```

Output:

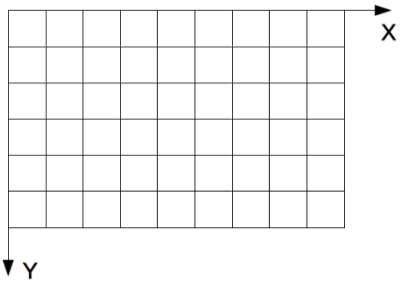
```
test 1
['001', '010', '011', '100', '101', '110', '111', '000', '001']

test 2
['001', '010', '011', '000', '001', '010', '011', '100', '101']

test 3
['001', '010', '011', '000', '001', '010', '000', '001', '010']
```

Part C

In the following problems, you are asked to do different parts of a computer game called DW2 (Digital World Warrior). DW2 requires two main classes, an Avatar (Q5) and a Map (Q6). **If you cannot finish these two questions, you can download the compiled file dw2lib.pyc from Tutor and import these Classes to do the subsequent questions. Moreover, you can also download all the test cases from Tutor**. The position coordinates in this game are shown in the picture below.



Q.5 [15 points]

Write a class called **Avatar** with the following details:

- It has three attributes called **name**, **hp** (for health point or HP), and **position**. The attribute **name** is of the type string, **hp** is an integer, and **position** is a tuple of two integers x and y.
- **def __init__()** : It initializes **name**, **hp**, and **position**. The argument to initialize **name** is required to create the object. If no arguments are given, **hp** has a default value of 100 and **position** has a default value of (1,1).
- Create the "getter" and "setter" methods for the three attributes, i.e. **getName**, **setName**, **getHP**, **setHP**, and **getPosition** and **setPosition**.
- **def heal()**: it takes one positive number as argument which is the amount of health points to be added to **hp**. If the argument is not specified, the default value is to heal one **hp**. If the argument is negative, it has no effect on the **hp**.
- **def attacked()**: it takes one negative number as argument which is the amount of **hp** to be modified when there is an attack. If the argument is not specified, the default value is -1 , which means that the **hp** is deducted by one point. If the argument is positive number, it has no effect on **hp**.

See test cases on this page and the next.

Sample tests:

```
print 'test 1: __init__'
a=Avatar('John')
print (a.name, a.hp, a.position)

print 'test 2: __init__'
a=Avatar('Jane',150,(10,10))
print (a.name, a.hp, a.position)

print 'test 3: getName(), setName()'
a=Avatar('John')
a.setName('Jude')
print a.getName()
```

Output:

```
test 1: __init__
('John', 100, (1, 1))

test 2: __init__
('Jane', 150, (10, 10))

test 3: getName(), setName()
Jude
```

Sample tests:

```
print 'test 4: getPosition(), setPosition()'
a=Avatar('John')
a.setPosition((1,3))
print a.getPosition()

print 'test 5: getHP(), setHP()'
a=Avatar('John')
a.setHP(200)
print a.getHP()

print 'test 6: heal()'
a=Avatar('John')
a.heal(5)
print a.getHP()

print 'test 7: attacked()'
a=Avatar('John')
a.attacked(20)
print a.getHP()

print 'test 8: heal()'
a=Avatar('John')
a.heal()
print a.getHP()

print 'test 9: attacked()'
a=Avatar('John')
a.attacked()
print a.getHP()

print 'test 10: heal(), attacked() '
a=Avatar('John')
a.attacked(2)
a.heal(-2)
print a.getHP()
```

Output:

```
test 4: getPosition(), setPosition()
(1, 3)

test 5: getHP(), setHP()
200

test 6: heal()
105

test 7: attacked()
100

test 8: heal()
101

test 9: attacked()
99

test 10: heal(), attacked()
100
```

Q.6 [15 points]

Write a class called **Map** with the following details:

- **def __init__()**: it takes one argument which is a dictionary that gives a map informatio. The initializer initializes an attribute called **world** using the argument. The attribute **world**, then, is a dictionary and its key is a position in the map and its value is either the food energy (if positive), enemy attack value (if negative) or other information like a wall (if it is an integer 0) or an exit point (if it is a character 'x'). See the test cases below for an example on the format of the dictionary. You need to avoid the problem of aliasing during this step.
- **def whatIsAt()**: it takes one argument which is a tuple of two integers indicating a position. The function checks what the map indicates at that particular position. There are several possible values in the **world** dictionary map:
    - if it is a character 'x', the function should return a string 'Exit',
    - if it is an integer 0, the function should return a string 'Wall',
    - if it is a positive integer, the function should return a string 'Food',
    - if it is a negative integer, the function should return a string 'Enemy',
    - if it is not found in the **world** dictionary, the function should return a string 'Empty'.

- **def getEnemyAttack()**: it takes one argument which is a tuple of two integers indicating a position. The function returns the damaged value affecting the **hp** if there is an 'Enemy' in that particular position. If there is no enemy in that position, it should return a Boolean False.
- **def getFoodEnergy()**: it takes one argument which is a tuple of two integers indicating a position. The function returns the energy value affecting the **hp** if there is a 'Food' in that particular position. If there is no enemy in that position, it should return a Boolean False.
- **def removeEnemy()**: it takes one argument which is a tuple of two integers indicating a position. The function checks if an 'Enemy' is in that position. If it is, it removes the 'Enemy' from the map by deleting the key-value pair in the dictionary world, and return a Boolean True. Otherwise, it will return a Boolean False.
- **def eatFood()**: it takes one argument which is a tuple of two integers indicating a position. The function checks if a 'Food' is in that position. If it is, it removes the 'Food' from the map by deleting the key-value pair in the dictionary world, and return a Boolean True. Otherwise, it will return a Boolean False.
- **def getExitPosition()**: it takes no input argument and returns either a tuple or a None object. If there is an 'Exit' in the map, the function returns the position as a tuple of two integers (x,y). Otherwise, it will return a None object. There is only one exit point in a map.

See test cases on this page and the next.

Sample tests:

```
world={(0,0):0, (1,0):0 , (2,0):0, (3,0): 0, (4,0):0, (5,0): 0,
(0,1):0, (1,1): 2, (2,1):-3, (5,1): 0, (0,2):0, (5,2): 0, (0,3):0,
(5,3): 0, (0,4):0, (5,4): 0, (0,5):0, (1,5):0 , (2,5):0, (3,5): 0,
(4,5):'x', (5,5): 0}

print 'test 1: object instantiation'
m=Map(world)
print m.world

print 'test 2: whatIsAt'
print m.whatIsAt((1,0))

print 'test 3: whatIsAt'
print m.whatIsAt((2,1))

print 'test 4: whatIsAt'
print m.whatIsAt((1,1))

print 'test 5: getFoodEnergy'
w1=m.getFoodEnergy((1,1))
w2=m.getFoodEnergy((3,3))
print (w1,w2)

print 'test 6: getEnemyAttack'
w1=m.getEnemyAttack((2,1))
w2=m.getEnemyAttack((3,3))
print (w1,w2)

print 'test 7: removeEnemy'
w1=m.getEnemyAttack((2,1))
w2=m.removeEnemy((2,1))
w3=m.getEnemyAttack((2,1))
print (w1,w2,w3)
```

Output:

```
test 1: object instantiation
{(3, 0): 0, (2, 1): -3, (5, 1): 0, (2, 5): 0, (0, 3): 0, (4, 0): 0,
(5, 5): 0, (1, 5): 0, (5, 0): 0, (0, 4): 0, (5, 3): 0, (1, 1): 2, (5,
4): 0, (0, 0): 0, (4, 5): 'x', (0, 5): 0, (1, 0): 0, (3, 5): 0, (0,
1): 0, (2, 0): 0, (5, 2): 0, (0, 2): 0}

test 2: whatIsAt
Wall

test 3: whatIsAt
Enemy
test 4: whatIsAt
Food

test 5: getFoodEnergy
(2, False)

test 6: getEnemyAttack
(-3, False)

test 7: removeEnemy
(-3, True, False)
```

Sample tests:

```
print 'test 8: whatIsAt'
print m.whatIsAt((1,4))

print 'test 9: getFoodEnergy'
print m.getFoodEnergy((1,4))

print 'test 10: getEnemyAttack'
print m.getEnemyAttack((1,4))

print 'test 11: whatIsAt'
print m.whatIsAt((4,5))

print 'test 12: getExitPosition'
print m.getExitPosition()

print 'test 13: eatFood'
w1=m.whatIsAt((1,1))
w2=m.eatFood((1,1))
w3=m.whatIsAt((1,1))
print (w1,w2,w3)

print 'test 14: test aliasing'
print m.world == world
```

Output:

```
test 8: whatIsAt
Empty

test 9: getFoodEnergy
False

test 10: getEnemyAttack
False

test 11: whatIsAt
Exit

test 12: getExitPosition
(4, 5)

test 13: eatFood
('Food', True, 'Empty')

test 14: test aliasing
False
```

Q.7 [Total: 20 points]

This question requires you to complete Q5 and Q6. If you have not finished those questions, use the provided compiled python file to import the Avatar and Map classes. You can find the compiled python file from Tutor. To import, save the file in your working directory and type the following line:

```
from dw2lib import Avatar, Map
```

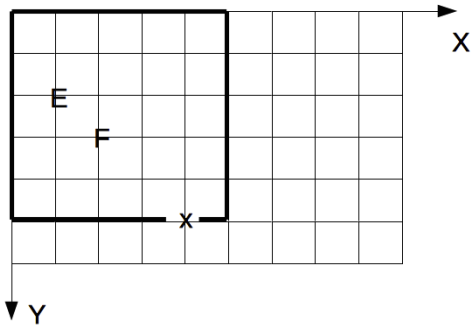
Create a state machine class called **DW2Game** as a child class of **sm.SM**. It has the following details:

- **def __init__()**: it takes two arguments, the first one is an object of type Avatar (Q5), and the second one is an object of type Map (Q6). This method initializes the attribute called **startState** as a tuple of these two objects, i.e. Avatar and Map, where the Avatar is the first element of the tuple and the Map is the second element. You need to avoid the problem of aliasing at this point.
- **def getNextValues()**: it takes two arguments which are the current state and the input. The function returns the next state and the output. This method should not modify the attribute **state** directly. This means that any changes to the state must be done by returning the next state. This function has the following details:
    - The input is a tuple of two strings. The first item is the mode which can only either be a ‘move’ or an ‘attack’. The second item is the direction of the of the mode. The ‘move’ mode can either be ‘up’, ‘right’, ‘down’ or ‘left’. Similarly, the ‘attack’ mode can either be ‘up’, ‘right’, ‘down’, or ‘left’.
    - The current state is a tuple of an Avatar object and a Map object. The next state is also a tuple of the Avatar object and the Map object. In the getNextValues, you should update the Avatar and the Map object and pass it as the next state. You should avoid aliasing in your code.

- The Avatar object can only move if it is neither a ‘Wall’ nor an ‘Enemy’. If it is an ‘Empty’ space, the getNextValues should update the position of the Avatar object to its new position. If it is a ‘Food’, the getNextValues should increase the hp of the Avatar according to the Food energy and update the position of the Avatar. Moreover, it should remove the food from the map by calling **eatFood()**.
- If the new position is a ‘Wall’, the Avatar object remains in its original position. If the new position is an ‘Enemy’, the Avatar object is attacked and its **hp** decreases according to the damaged point of the enemy stated in the map. The Avatar object also stays in its original position.
- To pass through an ‘Enemy’, the Avatar object must ‘attack’ the Enemy. When the ‘Enemy’ is attacked, it is removed from the map by calling **removeEnemy()** and the Avatar object can move to that position in the next time step.
- The output of the state machine is a tuple of three items. The first one is the name of the Avatar, the second one is the new position of the Avatar, and the last one is the **hp** of the Avatar.

- **def done()**: it takes one argument which is the current state. The function checks if the Avatar’s position is the same as the ‘Exit’ position. If they are the same, the function returns a Boolean True. Otherwise, it returns a Boolean False. This function is used by the SM class to terminate the state machine.

  To test, we use the following map, where 'x' indicates the 'Exit' point at (4,5), 'E' indicates the enemy at (1,2), and 'F' indicates 'Food' at (2,3).



See test cases on this page and the next.

Sample Tests:

```
world2={(0,0):0, (1,0):0 , (2,0):0, (3,0): 0, (4,0):0,
(5,0): 0, (0,1):0, (5,1): 0, (0,2):0, (1,2): -2, (5,2): 0
(0,3):0, (2,3): 3, (5,3): 0, (0,4):0, (5,4): 0, (0,5):0,
(1,5):0, (2,5):0, (3,5): 0, (4,5):'x', (5,5): 0,}

print 'test 1'
inp=[('move','down'),('attack','down'),('move','down'),(
'move','down'),
('move','down'),('move','right'),('move','right'),('move','
right'),('move','down'),
('move','down')]
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
print g.transduce(inp)
```

Output:

```
test 1
[('John', (1, 1), 98), ('John', (1, 1), 98), ('John', (1, 2), 98),
('John', (1, 3), 98), ('John', (1, 4), 98), ('John', (2, 4), 98),
('John', (3, 4), 98), ('John', (4, 4), 98), ('John', (4, 5), 98)]
```

Sample Tests:

```
print 'test 2'
inp=[('move','left'),('move','right'),('move','right'),('mov
e','right'),
('move','right'),('move','down'),('move','down'),('move'
,'down'),
('move','up')]
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
print g.transduce(inp)

print 'test 3'
inp=[('move','right'),('move','right'),('move','right'),
('move','down'),('move','left'),('move','left'),('move','lef
t'),
('attack','left'),('move','left')]
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
print g.transduce(inp)

print 'test 4'
inp=[('move','right'),('move','right'),('move','right'),
('move','down'),('move','left'),('move','left'),('move','lef
t'),
('attack','left'),('move','left'),('move','left'),('move','dow
n'),
('move','right')]
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
print g.transduce(inp)

print 'test 5'
inp=[('move','right'),('move','right'),('move','right'),
('move','down'),('move','left'),('move','left'),('move','lef
t'),
('attack','left'),('move','left'),('move','left'),('move','dow
n'),
('move','right'),('move','right'),('move','right'),('move','
down'),('move','down'),
('move','down')]
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
print g.transduce(inp)


print 'test 6'
av=Avatar('John')
m=Map(world2)
g=DW2Game(av,m)
g.start()
n,o=g.getNextValues(g.startState,('move','right'))
ans = g.state[0].getPosition() == n[0].getPosition()
print ans, g.state[0].getPosition(), n[0].getPosition()
```

Output:

```
test 2
[('John', (1, 1), 100), ('John', (2, 1), 100), ('John', (3, 1),
100), ('John', (4, 1), 100), ('John', (4, 1), 100), ('John', (4,
2), 100), ('John', (4, 3), 100), ('John', (4, 4), 100), ('John',
(4, 3), 100)]

test 3
[('John', (2, 1), 100), ('John', (3, 1), 100), ('John', (4, 1),
100), ('John', (4, 2), 100), ('John', (3, 2), 100), ('John', (2,
2), 100), ('John', (2, 2), 98), ('John', (2, 2), 98), ('John', (1,
2), 98)]

test 4
[('John', (2, 1), 100), ('John', (3, 1), 100), ('John', (4, 1),
100), ('John', (4, 2), 100), ('John', (3, 2), 100), ('John', (2,
2), 100), ('John', (2, 2), 98), ('John', (2, 2), 98), ('John', (1,
2), 98), ('John', (1, 2), 98), ('John', (1, 3), 98), ('John', (2,
3), 101)]

test 5
[('John', (2, 1), 100), ('John', (3, 1), 100), ('John', (4, 1),
100), ('John', (4, 2), 100), ('John', (3, 2), 100), ('John', (2,
2), 100), ('John', (2, 2), 98), ('John', (2, 2), 98), ('John', (1,
2), 98), ('John', (1, 2), 98), ('John', (1, 3), 98), ('John', (2,
3), 101), ('John', (3, 3), 101), ('John', (4, 3), 101), ('John',
(4, 4), 101), ('John', (4, 5), 101)]

test 6
False (1,1) (2,1)
```

Part D (Optional)

Q.8 [8 points]

Modify the class definition for Map (Q6) to add the following method:

- **def getSearchMap():** this method takes no input argument and returns a dictionary. The function makes use of the attribute **world**, the map dictionary, that contains positions of the 'Wall', 'Enemy', 'Food', and 'Exit', and generates a new map dictionary that contains all the positions in the map and their relationship with surrounding positions. The output dictionary has the following characteristics:
    - The keys are tuples of two integers indicating positions. The map is always a rectangle and the keys contains all the (x,y) positions of the map. For example, the sample test case has a top left coordinate at (0,0) and a bottom right coordinate at (5,5). Therefore, the function will generate a dictionary with keys of all the possible points from (0,0) to (5,5). See test cases below for an example.
    - The values are dictionaries showing the cost of connection from the position indicated by the key to its neighbouring positions. An example of a key-value pair is shown here.

      (1, 3): {0: ((1, 2), **5**), 1: ((2, 3), **0**), 2: ((1, 4), **3**), 3: ((0, 3), **1000**)}

      In this case, the key is the point at (1,3). This point has four neighbouring points. They are at (1,2), (2,3), (1,4), and (0,3), which corresponds to the 'top' (key is 0), 'right' (key is 1), 'bottom' (key is 2), and 'left' (key is 3) points of position (1,3). Notice that the keys for these neighbouring points are the direction from point (1,3).
      In each element, the first item is a tuple of two integers indicating the position of the surrounding location. The second item is an integer indicating the cost of moving to that neighbouring position. In the above example:
        - moving from (1,3) to (1,2) has a cost of **5**
        - moving from (1,3) to (2,3) has a cost of **0**
        - moving from (1,3) to (1,4) has a cost of **3**
        - moving from (1,3) to (0,3) has a cost of **1000**

      Notice that:
        - If the key indicates a position at a corner, it only has two neighbouring points and, thus, the value is a dictionary of two items.
        - If the key indicates a position at the boundary but not a corner, it has three neighbouring points, and thus, the value is a dictionary of three items.
        - If the key indicates a position not at the boundary, it has four neighbouring points, and thus, the value is a dictionary of four items.

    - The cost is generated as follows:
        - Cost of moving to a position is a non-negative number.
        - An offset is determined by the maximum non-negative value in the original dictionary map. This is determined by the energy point of 'Food' with a

maximum value in the map. For example, in the world test case 3 we use below, there is a 'Food' at (2,3) with a point of 3. This is the maximum non-negative value, and therefore, the offset is 3.

- If the neighbouring position is of type 'Enemy', the cost is absolute value of the damaged point plus the offset. For example, in the world test case 3 we use below, there is an 'Enemy' at (1,2) with a value of -2. Therefore, the cost is 2+3 = 5. See example above for the cost of moving from (1,3) to (1,2).
- The cost of a 'Food' is the negative value of the energy point plus the offset. For example, in the world test case 3 we use below, there is a 'Food' with a value 3 and the offset of the world is 3. Therefore, the cost will be -3+3=0. The maximum energy 'Food' will have a cost of 0. If there is a 'Food' with 2 energy point, the cost will be -2 + 3 = 1. See example above for the cost of moving from (1,3) to (2,3).
- The cost of moving to a 'Wall' is fixed as 1000. See example above for the cost of moving from (1,3) to (0,3).
- The cost of moving to an 'Empty' space is simply the offset. See example above for the cost of moving from (1,3) to (1,4).

Sample Tests:

```
world={(0,0):0, (1,0):0 , (2,0):0, (0,1):0, (1,1):-2, (2,1): 0, (0,2):0, (1,2): 'x', (2,2): 0}

print 'test 1'
m=Map(world)
print m.getSearchMap()

world={(0,0):0, (1,0):0 , (2,0):0, (0,1):0, (1,1):3, (2,1): 0, (0,2):0, (1,2): 'x', (2,2): 0}

print 'test 2'
m=Map(world)
print m.getSearchMap()

world={(0,0):0, (1,0):0 , (2,0):0, (3,0): 0, (4,0):0, (5,0): 0, (0,1):0, (5,1): 0, (0,2):0, (1,2): -2, (5,2): 0, (0,3):0, (2,3): 3,
(5,3): 0, (0,4):0, (5,4): 0, (0,5):0, (1,5):0 , (2,5):0, (3,5): 0, (4,5):'x', (5,5): 0}

print 'test 3'
m=Map(world)
print m.getSearchMap()
```

Output:

```
test 1

{(0, 1): {0: ((0, 0), 1000), 1: ((1, 1), 2), 2: ((0, 2), 1000)}, (1, 2): {0: ((1, 1), 2), 1: ((2, 2), 1000), 3: ((0, 2), 1000)}, (0,
0): {1: ((1, 0), 1000), 2: ((0, 1), 1000)}, (2, 1): {0: ((2, 0), 1000), 2: ((2, 2), 1000), 3: ((1, 1), 2)}, (1, 1): {0: ((1, 0),
1000), 1: ((2, 1), 1000), 2: ((1, 2), 0), 3: ((0, 1), 1000)}, (2, 0): {2: ((2, 1), 1000), 3: ((1, 0), 1000)}, (2, 2): {0: ((2, 1),
1000), 3: ((1, 2), 0)}, (1, 0): {1: ((2, 0), 1000), 2: ((1, 1), 2), 3: ((0, 0), 1000)}, (0, 2): {0: ((0, 1), 1000), 1: ((1, 2), 0)}}

test 2

{(0, 1): {0: ((0, 0), 1000), 1: ((1, 1), 0), 2: ((0, 2), 1000)}, (1, 2): {0: ((1, 1), 0), 1: ((2, 2), 1000), 3: ((0, 2), 1000)}, (0,
0): {1: ((1, 0), 1000), 2: ((0, 1), 1000)}, (2, 1): {0: ((2, 0), 1000), 2: ((2, 2), 1000), 3: ((1, 1), 0)}, (1, 1): {0: ((1, 0),
1000), 1: ((2, 1), 1000), 2: ((1, 2), 3), 3: ((0, 1), 1000)}, (2, 0): {2: ((2, 1), 1000), 3: ((1, 0), 1000)}, (2, 2): {0: ((2, 1),
1000), 3: ((1, 2), 3)}, (1, 0): {1: ((2, 0), 1000), 2: ((1, 1), 0), 3: ((0, 0), 1000)}, (0, 2): {0: ((0, 1), 1000), 1: ((1, 2), 3)}}
```

test 3

{(1, 3): {0: ((1, 2), 5), 1: ((2, 3), 0), 2: ((1, 4), 3), 3: ((0, 3), 1000)}, (3, 0): {1: ((4, 0), 1000), 2: ((3, 1), 3), 3: ((2, 0), 1000)}, (5, 4): {0: ((5, 3), 1000), 2: ((5, 5), 1000), 3: ((4, 4), 3)}, (2, 1): {0: ((2, 0), 1000), 1: ((3, 1), 3), 2: ((2, 2), 3), 3: ((1, 1), 3)}, (5, 1): {0: ((5, 0), 1000), 2: ((5, 2), 1000), 3: ((4, 1), 3)}, (2, 5): {0: ((2, 4), 3), 1: ((3, 5), 1000), 3: ((1, 5), 1000)}, (0, 3): {0: ((0, 2), 1000), 1: ((1, 3), 3), 2: ((0, 4), 1000)}, (4, 0): {1: ((5, 0), 1000), 2: ((4, 1), 3), 3: ((3, 0), 1000)}, (1, 2): {0: ((1, 1), 3), 1: ((2, 2), 3), 2: ((1, 3), 3), 3: ((0, 2), 1000)}, (3, 3): {0: ((3, 2), 3), 1: ((4, 3), 3), 2: ((3, 4), 3), 3: ((2, 3), 0)}, (4, 4): {0: ((4, 3), 3), 1: ((5, 4), 1000), 2: ((4, 5), 3), 3: ((3, 4), 3)}, (1, 5): {0: ((1, 4), 3), 1: ((2, 5), 1000), 3: ((0, 5), 1000)}, (5, 0): {2: ((5, 1), 1000), 3: ((4, 0), 1000)}, (2, 2): {0: ((2, 1), 3), 1: ((3, 2), 3), 2: ((2, 3), 0), 3: ((1, 2), 5)}, (5, 3): {0: ((5, 2), 1000), 2: ((5, 4), 1000), 3: ((4, 3), 3)}, (4, 1): {0: ((4, 0), 1000), 1: ((5, 1), 1000), 2: ((4, 2), 3), 3: ((3, 1), 3)}, (1, 1): {0: ((1, 0), 1000), 1: ((2, 1), 3), 2: ((1, 2), 5), 3: ((0, 1), 1000)}, (3, 2): {0: ((3, 1), 3), 1: ((4, 2), 3), 2: ((3, 3), 3), 3: ((2, 2), 3)}, (0, 0): {1: ((1, 0), 1000), 2: ((0, 1), 1000)}, (4, 5): {0: ((4, 4), 3), 1: ((5, 5), 1000), 3: ((3, 5), 1000)}, (0, 4): {0: ((0, 3), 1000), 1: ((1, 4), 3), 2: ((0, 5), 1000)}, (5, 5): {0: ((5, 4), 1000), 3: ((4, 5), 3)}, (1, 4): {0: ((1, 3), 3), 1: ((2, 4), 3), 2: ((1, 5), 1000), 3: ((0, 4), 1000)}, (0, 5): {0: ((0, 4), 1000), 1: ((1, 5), 1000)}, (4, 2): {0: ((4, 1), 3), 1: ((5, 2), 1000), 2: ((4, 3), 3), 3: ((3, 2), 3)}, (1, 0): {1: ((2, 0), 1000), 2: ((1, 1), 3), 3: ((0, 0), 1000)}, (3, 5): {0: ((3, 4), 3), 1: ((4, 5), 3), 3: ((2, 5), 1000)}, (0, 1): {0: ((0, 0), 1000), 1: ((1, 1), 3), 2: ((0, 2), 1000)}, (5, 2): {0: ((5, 1), 1000), 2: ((5, 3), 1000), 3: ((4, 2), 3)}, (3, 1): {0: ((3, 0), 1000), 1: ((4, 1), 3), 2: ((3, 2), 3), 3: ((2, 1), 3)}, (0, 2): {0: ((0, 1), 1000), 1: ((1, 2), 5), 2: ((0, 3), 1000)}, (2, 0): {1: ((3, 0), 1000), 2: ((2, 1), 3), 3: ((1, 0), 1000)}, (4, 3): {0: ((4, 2), 3), 1: ((5, 3), 1000), 2: ((4, 4), 3), 3: ((3, 3), 3)}, (2, 3): {0: ((2, 2), 3), 1: ((3, 3), 3), 2: ((2, 4), 3), 3: ((1, 3), 3)}, (3, 4): {0: ((3, 3), 3), 1: ((4, 4), 3), 2: ((3, 5), 1000), 3: ((2, 4), 3)}, (2, 4): {0: ((2, 3), 0), 1: ((3, 4), 3), 2: ((2, 5), 1000), 3: ((1, 4), 3)}}

Q.9 [Bonus point: 7 points]

Write a function called **findPath** that takes in two arguments: an Avatar, and a Map. The function returns a list of tuples indicating the movements to take from the Avatar's position to an exit point in the Map and the total cost to move to that exit point. The possible list of movement is:

- 0 means move 'up'
- 1 means move 'right'
- 2 means move 'down'
- 3 means move 'left'

In the sample test 1 on the next page, the Avatar starts at (1,3) and an exit point is found at (4,5). The output path then moves the avatar to the 'right' three times and then moves it down two times. If there is no possible path to reach the exit point, the function should return **None**. If there are multiple paths you should return the one with the minimum cost.

*Hint: You can use **getSearchMap()** method to generate all possible paths with their corresponding costs. You can then write a search function to find the path with the minimum cost or you may want to explore the module in **libdw** called **ucSearch**. See documentation in Tutor.*

See test cases on the next page.

Sample Tests:

world={(0,0):0, (1,0):0, (2,0):0, (3,0): 0, (4,0):0,
(5,0): 0, (0,1):0, (5,1): 0,
(0,2):0, (1,2): -2, (5,2): 0, (0,3):0, (2,3): 3, (5,3): 0,
(0,4):0, (5,4): 0,

```
(0,5):0, (1,5):0, (2,5):0, (3,5): 0, (4,5):'x', (5,5): 0}

print 'test 1'
av=Avatar('John',position=(1,3))
m=Map(world)
print findPath(av,m)

world={(0,0):0, (1,0):0, (2,0):0, (3,0): 0, (4,0):0,
(5,0): 0, (0,1):0, (5,1): 0, (0,2):0, (1,2): -2, (5,2): 0,
(0,3):0, (2,3): 3, (3,3):0, (5,3): 0, (0,4):0, (3,4):0,
(5,4): 0, (0,5):0, (1,5):0, (2,5):0, (3,5): 0, (4,5):'x',
(5,5): 0}

print 'test 2'
av=Avatar('John',position=(1,3))
m=Map(world)
print findPath(av,m)

world={(0,0):0, (1,0):0 , (2,0):0, (3,0): 0, (4,0):0,
(5,0): 0,(0,1):0, (3,1):0, (5,1): 0,(0,2):0, (1,2): -2,
(3,2):0, (5,2): 0,(0,3):0, (2,3): 3, (3,3):0, (5,3):
0,(0,4):0, (3,4):0, (5,4): 0,(0,5):0, (1,5):0 , (2,5):0,
(3,5): 0, (4,5):'x', (5,5): 0}

print 'test 3'
av=Avatar('John',position=(1,3))
m=Map(world)
print findPath(av,m)
```

Output:

```
test 1
([(None, (1, 3)), (1, (2, 3)), (1, (3, 3)), (1, (4, 3)),
(2, (4, 4)), (2, (4, 5))], 12)

test 2
([(None, (1, 3)), (1, (2, 3)), (0, (2, 2)), (1, (3, 2)),
(1, (4, 2)), (2, (4, 3)), (2, (4, 4)), (2, (4, 5))], 18)

test 3
None
```

End of Exam Paper

[this page is left blank]