MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 111

Fall 2020

Take-Home Exam 2

REGULATIONS

Due date: 13 January 2021, Sunday, 23:59 *(Not subject to postpone)*

Submission: Electronically. You will be submitting your program source code through a text file which you will name as `the2.py` by means of the CENGCLASS system. Details will be announced on the forum.

Team: There is **no** teaming up. The homework has to be done and turned in individually.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

This exam is about modelling the spread of a contagious disease. Individuals are moving randomly obeying a probabilistic rule and may get in the vicinity of infecting some other individual(s) with some probability. The movements are not intelligent though physical.

TASK DESCRIPTION

There are P individuals, $I_0, ..., I_{P-1}$, each occupying a cell of an $M \times N$ grid ($M\,(rows) \leq 100$; $N(columns) \leq 100$). Distance is measured in the conventional way: Euclidean distance. Individuals move from one cell to a neighbouring cell with a probability explained below. There are 8 possible neighbouring cells. The moves are made in the order of the individuals (you can assume that individuals are numbered). For each individual, his/her move depends on his/her previous move and the 'probability' (explained below) and nothing else.

In one unit of time, after all the moves are performed, the rules of *contamination* apply: With a probability which is inversely proportional to the distance between two individuals, if one of the individuals is infected, and the other is not, he/she can transmit the disease to the other individual. This probability is effective only if the distance between them is under a threshold value D. If an individual gets infected, he/she stays infected (i.e. there is no cure) and starts to be infectious immediately in the next time frame (yes, a little bit unrealistic).

The mutual infection probability of two individuals I_i, I_j is as follows:

$$\mathcal{P}_{ij} = \begin{cases} \min\left(1, \dfrac{\kappa}{distance_{ij}^2}\right), & \text{if } distance_{ij} \leq D \\ 0, & \text{otherwise} \end{cases}$$

Individuals may wear masks which reduce the infection probability by a factor of λ. This is explained in detail below.

M, N, K, λ, D are system constants and are defined in the Specifications section.

The move probabilities are set according to the last move. In Figure 1, two possible last moves are indicated as red arrows. The left one is one of a forward/backward/left/right move. To understand it, align the last move arrow of an individual with the red arrow in the left picture. In the right picture, the last move was a

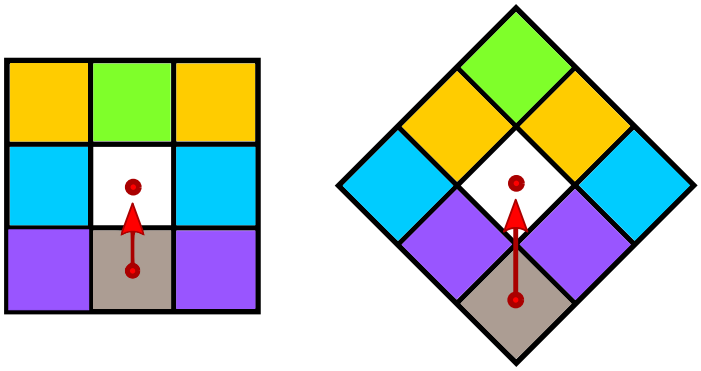


Figure 1: Illustration of movement probabilities. See the text for details.

Table 1: The colors and the associated probabilities.

GREEN	$\frac{1}{2}\mu$
YELLOW	$\frac{1}{8}\mu$
BLUE	$\frac{1}{2}\left[1-\mu-\mu^2\right]$
PURPLE	$\frac{2}{5}\mu^2$
GRAY	$\frac{1}{5}\mu^2$

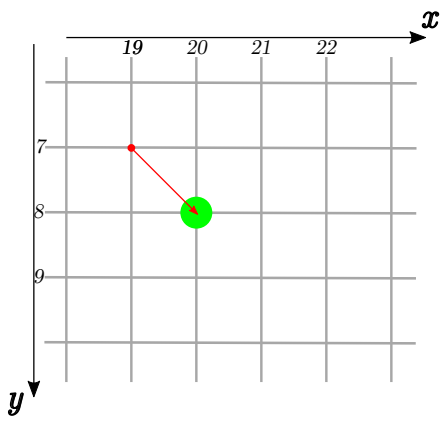
diagonal one. It was one of the forward-left/forward-right/backward-left/backward-right. Again, understand it by aligning the last move arrow with the red arrow.

The probabilities for the next move are indicated by the colors and are defined in Table 1 (μ is a constant and is $(0.1 \leq \mu \leq 0.6)$):

The initial state of the individuals, including a hypothetical last move information, will be provided as part of the initial state – you will get it by calling a function, at the start of your program.

Each movement of the individuals is instantaneous. There is a strict order to be followed in the evaluation of the next move, as you will see in the Specifications section. This is to be able to provide for every student the probabilities in the same sequence (it has actually nothing to do with the dynamics of the system) and impose a proper decision order for 'attempting preoccupied positions' cases. *(This is something that eases the task. Otherwise an intermediate recording of the universe state had to be made and additional 'who will take the position in double attempts' rules had to be implemented (hadi gine iyisiniz!))*

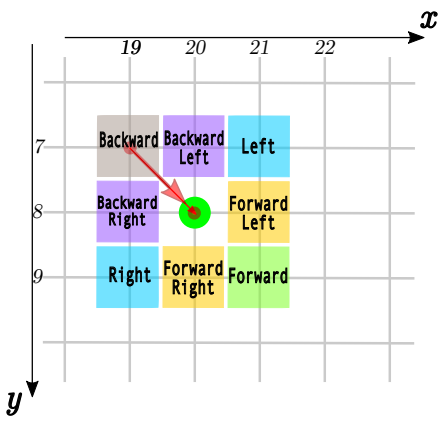
Below you see an example of a single move:



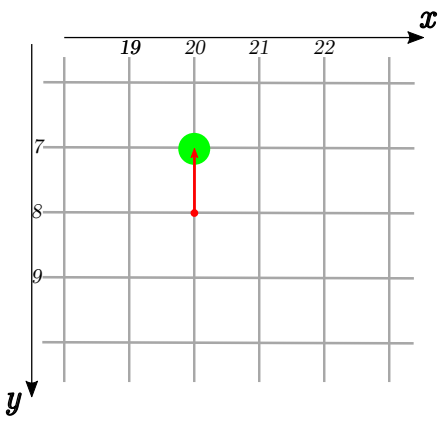
A `"notmasked"`, `"notinfected"` individual is at arena position `(20,8)`. He/she arrived here from `(19,7)`. This would be given as a state information as:

```
[(20,8), 7,"notmasked",
  "notinfected"]
```

The `7`, the second element in the list, denotes that the last move was in direction 7.



We align the red arrow of the probability template given in Figure 1 with the last move arrow of the individual. The relative move probabilities (relative to the red arrow) are given (see Table 1) (μ is a number provided in data obtained by the `get_data()` call). Now it is time to make a probabilistic draw among these 8 relative directions, with the given move probabilities.



Let us assume, as an example, the probabilistic draw which you made according to these set of probabilities returned to you `"Backward-Left"`. Then the individual moves to `(20,7)`. Now its state information reads as:

```
[(20,7), 4,"notmasked",
  "notinfected"]
```

The `4`, the second element in the list, denotes that the last move was in coordinate-wise direction 4 (see Table 2).

In case of attempting a move to a preoccupied position or outside of the arena, the attempting individual will wait for one turn and hope the probability in the next turn will assign him/her to an empty neighbour square that time (do not think of performing an immediate renewed probabilistic move: he/she should wait the next unit of time). It is possible (with low probability) that an individual gets trapped and cannot move at all, for more then one unit of time.

Individuals are set up as wearing a mask or not. This does not change over time in the simulation. If an individual is wearing a mask, his/her probability to get infected or transmit his/her infection to another individual is reduced by a dividing factor of λ. If the interaction is among two mask wearers, then the probability is reduced by dividing it by a factor of λ^2. $(1 < \lambda)$ and is (realistically) around multiples of ten.

For each time step of the simulation, a function that you will code will be called by the evaluating script and will be expected to return the new coordinates (that you have calculated) for each individual (made into a list).

We know how valuable visualisation is. So, we provide you with a code that visualizes your results. This is not part of the grading process.

Also another program with a GUI is provided to visually construct some input data (for development and testing purposes only). The data this program generates is **not** going to be submitted.

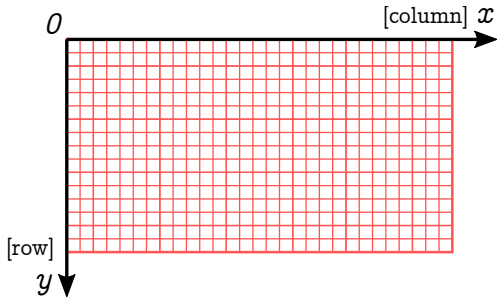


Figure 2: The arena and how the coordinates change.

Table 2: Coordinate-wise `last_move` direction labels

`last_move`	**Coordinate-wise direction**
0	$(+y, 0)$
1	$(+y, -x)$
2	$(0, -x)$
3	$(-y, -x)$
4	$(-y, 0)$
5	$(-y, +x)$
6	$(0, +x)$
7	$(+y, +x)$

SPECIFICATIONS

- The coordinate system on the arena where the individuals move (we call it also as *universe*) is illustrated in Figure 2.
- There will be at least 1 individual and at most 100 individuals ($1 \leq P \leq 100$).
- A `universal_state` is represented as a list of P individuals where the ith position holds the state of the ith individual ($i \in [0, P-1]$). The *state of an individual* is represented in a list of the form:

  `[(x,y), last_move, mask_status, infection_status]`

  where the elements encode the following information:

  - `x` (column): an integer, $0 \leq x < N$
  - `y` (row): an integer, $0 \leq y < M$
  - `last_move`: an integer, $0 \leq$ `last_move` < 8. The semantics of each value is given in Table 2:
  - `mask_status`: `"masked"` or `"notmasked"`
  - `infection_status`: `"infected"` or `"notinfected"`
- To get your input data, you should execute a call to the function `get_data()` in `the2.py`. This call will return a list like:

  [M, N, D, κ, λ, μ, `universal_state`]

  The constants M, N, D, κ, λ, μ are explained in the TASK DESCRIPTION above. If needed at all, P is deducible since it is the member count of the `universal_state`.

  κ, λ, μ are floating point values. Multiple calls to `get_data()` in a single run will return the same values.

- In your `the2.py`, you are allowed to import only from:
  - the `math` module.
  - the `random` module.

  You cannot import form any other file/library.
- For each consecutive time frames you are expected to do the following in this order:
  1. for all $i \in [0, P-1]$, compute (probabilistically) the new positions for individual I_i.
  2. for all $i \in [0, P-1]$
     for all $j \in [i+1, P-1]$
     compute (probabilistically) new infection states for both I_i and I_j based on $I_i \leftrightarrow I_j$ interaction
- In order not to corrupt the evaluation process, do not alter the looping order. For the degenerate cases where both individuals have the same infection state, do not call a function from the `random` package. For the other cases, call such a function only once per interaction.
- Hint: Check out `random.choices()`. Find its definition. Study and understand it. We advise that you construct and run small test programs. Use this function for the probabilistic move computation and the probabilistic infection computation.
- You are expected to write a function that you will **exactly name as** `new_move()`, which will take no arguments and (at each call) return the next time frame's `universal_state`, a list of the structure:

```
[[(x0, y0), last_move0, mask_status0, infection_status0],
     [(x1, y1), last_move1, mask_status1, infection_status1],
     ... ]
```

  The evaluators will be able to observe the whole event simulation by successive calls to your `new_move()`.
- `universal_state` contains also the mask information. It is obvious that this is not going to change over the simulation (it is set in the universe provided in the `get_data()` call), still it is going to be kept in the `universal_state` you will return per `new_move()` call. We keep it there for programming simplicity.
- To test your own code, you will need a `get_data()` function. For that purpose, you should construct your own `get_data()` function. We suggest that you write the code that defines your `get_data()` into a file named `evaluator.py`, which will be automatically loaded by the `draw.py` package.
  - Do not define this `get_data()` (that you will be using for your own testing) in the `the2.py` file.
  - Do not submit your `evaluator.py` file or any other file except `the2.py`. As evaluators, we are not interested in it. We will implement our own `evaluator.py` (which will implement our way of a `get_data()`
- Make sure that `new_move()` is implemented in `the2.py`, and conforms to the specifications. You are free to implement helper functions as much as you like. Moreover, you can (and probably will) use global variables. Do not execute a call of `new_move()` in `the2.py` since this is the job of the evaluators. You do not have any job other than providing a working `new_move()` function.
- Do not perform any error checks. The input will be error free.
- You can only use the concepts covered in the lectures until the deadline of THE2. That means no object-oriented programming solution.

HOW-TO-USE `draw.py`

The use of `draw.py` is optional. It serves only visualization purposes. Your code in `the2.py` will be evaluated using black-box batch testing and not visually.

- Place `the2.py`, `draw.py`, `evaluator.py` in your working directory and run the `draw.py` in Python (`draw.py` will import `the2.py` and `evaluator.py`).

  We provide sample `the2.py` and `evaluator.py` files with dummy `new_move()` and `get_data()` function definitions for your convenience. The sample `new_move()` does no calculation at all, and always returns a constant result; it is only for you to see how `draw.py` works.

- `draw.py` uses a coordinate system where the origin is at the top-left corner. x-axis is in the left-to-right direction and y-axis is in the top-to-bottom direction. So, it conforms to the coordinate definitions of this task.

- There is a single global variable used by `draw.py`, that you can change. If you want to alter its default value, do so by editing their assignment lines in `draw.py`.

  `DELAY:` The delay (time in milliseconds) between two successive `new_move()` calls. The default is 1000 (ms).

- To avoid name clashes, do not re-define something with the names of the global variables and functions in `draw.py`.

HOW-TO-USE `create_universe.py`

Watch the 12min. Youtube video announced in cengclass forum.

GRADING

- Comply with the specifications. Since your returned results will be evaluated automatically, non-compliant results will be considered as incorrect by our evaluation system.

- Your program will be tested with multiple data (a distinct run for each data).

- Any program that performs only 40% of the total grade or below will enter a glass-box test (eye inspection by the evaluator TA). The TA will judge an overall THE2 grade in the range of [0,40]. The glass-box test grade is not open to negotiation nor explanation.

We wish everyone good health amid the Covid-19 Pandemic.