Problem: PathDefense

Problem Statement

You are given a square map containing **N** by **N** cells. On the map you have **B** bases at fixed locations. A number of **P** paths lead from the boundary of the map towards your bases. Your bases are under attack and you need to defend them by placing defensive towers on the map. Luckily creeps only walk on the paths and can not attack your towers. You are rewarded with a fixed amount of money for each creep that your towers kill. You can build **T** types of towers, each tower will cost you money and can only attack creeps within a specific range and will do a fixed amount of damage when attacking the closest creep. Only one tower can be placed on each cell in the map. Multiple creeps can occupy the same cell. Creeps will spawn randomly on the boundary of the map and select random paths towards your bases. Additionally, at random times waves of creeps will rush towards some of your bases. A creep will do damage to your base when it reaches it and disappear after that (The amount of damage will be equal to the amount of health of the creep).

Your task is to maximize the sum of the amount of money you have and the total base health at the end of the simulation.

Implementation Details

Your code should implement the method `init(vector <string> board, int money, int creepHealth, int creepMoney, vector <int> towerType)`. Your `init` method will be called once, before the simulation starts. You can return any int from `init()` method, it will be ignored.

- **board** gives you the map containing **N** by **N** cells. Each string contains a row of the map. Any numeric character (**'0'**..**'9'**) denotes a base and it's index, **'.'** a path and **'#'** a cell where towers can be placed.
- **money** gives you the amount of money in hand at the start of the simulation.
- **creepHealth** gives you the amount of health points a creep will initially spawn with. Note that after every 500 simulation steps, the starting health of newly spawned creeps will double.
- **creepMoney** gives you the amount of money received for killing a creep.
- **towerType** gives you information about each type of tower that can be placed. Tower type **i** will shoot creeps within a range of `towerType[i*3]` cells and decrease the health of the creep by `towerType[i*3+1]` points. It will cost you `towerType[i*3+2]` to place the tower.

Your code should implement the method `placeTowers(vector <int> creep, int money, vector <int> baseHealth)`. Your method will be called **2000** times, once for every simulation step.

- **creep** gives you information about the current creeps on the map. Each creep is described by 4 values. Creep **i** is at column `creep[i*4+2]` and row `creep[i*4+3]`. `creep[i*4+1]` contains the health of the creep. `creep[i*4]` contains the unique creep id that can be used to track the creep movements.
- **money** gives you the current amount of money in hand.
- **baseHealth** gives you the health of your bases. `baseHealth[i]` contains the health of base **i**.

You must return the location and type of towers that you want to buy and place on the map. Each 3 elements of your return must describe one tower placement. Let vector <int> **ret** be your return, then **ret[j*3]** contains the column, **ret[j*3+1]** the row and **ret[j*3+2]** the tower type of your **j**-th tower to be placed during the current simulation step. You can only buy towers that you can afford with the current money in hand. Trying to buy a tower that you can not afford will result in a zero score. You don't have to place a tower at each step, you can return an empty array.

Scoring

Your score for an individual test case will be the sum of the amount of money in hand and base health at the end of the 2000 simulation steps. If your return has invalid format or specifies any invalid tower placements, your score for the test case will be 0. Your overall score will be calculated in the following way: for each test case where your score is not 0, you get 1 point for each competitor you beat on this test case (i.e., your score on a test case is larger than this competitor's score) and 0.5 points for each competitor you tie with (a tie with yourself is not counted); finally, the sum of points is divided by (the number of competitors - 1), then multiplied by 1000000 and divided by the number of test cases.

Clarifications

- There will be 2000 simulation steps for each test case.
- Each base will start with a health of 1000.
- Once a tower has been placed, it will stay there for the remaining part of the simulation. It can not be destroyed, moved or upgraded.
- Each tower will only attack once in each step and will only attack the nearest creep (if in range). If more than one creep is at the same nearest distance, the one with the lowest unique id will be attacked.
- If the tower is located at position (**X1**,**Y1**) and the range of a tower is **R**, then the tower can attack a creep at position (**X2**,**Y2**) only if **(X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)<=R*R**.
- Creeps will move one cell each simulation step.
- Creep will attack any base, even if the base has zero health.

- Creep will disappear after it attacked a base.
- After every 500 simulation steps, the starting health of newly spawned creeps will double.
- Multiple creeps can occupy the same cell on the map.
- Towers can only be placed on cells that are not a base or a path, at most one tower per cell.
- The amount of damage a creep will cause to a base is the creep's current health when reaching the base.
- The order of execution for each simulation step will be: Place your new towers, Creeps move and attack, your towers attack creeps.
- Your towers will attack in sequence, those that were build earlier will attack first. If a tower kills a creep, the next attacking tower will not try to attack the dead creep.
- The base health can't go lower than zero.

Test Case Generation

Please look at the visualizer source code for more detail about test case generation. Each test case is generated as follows:

- The map size **N** is randomly selected between 20 and 60, inclusive.
- The creep starting health is chosen between 1 and 20. The money reward for a creep is chosen between 1 and 20.
- The number of tower types **T** is randomly selected between 1 and 20, inclusive. For each tower type the range is chosen between 1 and 5. The damage done is chosen between 1 and 5. The cost is chosen between 5 and 40.
- The starting amount of money is equal to the sum of all tower type costs.
- The number of bases **B** is randomly selected between 1 and 8, inclusive. The location of each base is selected randomly and will be at least 4 cells away from the boundary of the map.
- The number of paths **P** is randomly selected between B and B*10. **P** random paths are then created between a random boundary location and random base.
- The number of total creeps **Z** is randomly selected between 500 and 2000. **Z** random creeps are created to spawn at specific simulation times at specific boundary points on paths.
- The number of waves **W** is randomly selected between 1 and 15. **W** waves of attacking creeps are selected to be spawned at the same boundary location. The wave of creeps will spawn closely together at a random time during the simulation.
- All values are chosen uniformly and independently, at random.

Tools

An offline tester/visualizer is available here. You can use it to test/debug your solution locally. You can also check its source code for exact implementation of test case generation, simulation and score calculation.

Definition

Class:	PathDefense
Method:	placeTowers
Parameters:	vector <int>, int, vector <int>
Returns:	vector <int>
Method signature:	vector <int> placeTowers(vector <int> creep, int money, vector <int> baseHealth)

Method:	init
Parameters:	vector <string>, int, int, int, vector <int>
Returns:	int
Method signature:	int init(vector <string> board, int money, int creepHealth, int creepMoney, vector <int> towerTypes)

(be sure your methods are public)

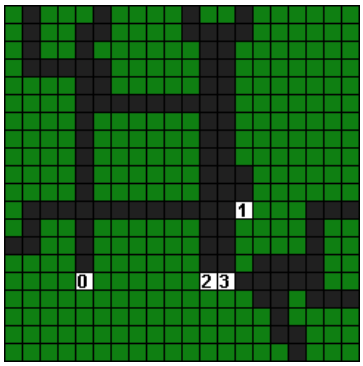
Notes

- The time limit is 20 seconds per test case (this includes only the time spent in your code). The memory limit is 1024 megabytes.
- There is no explicit code size limit. The implicit source code size limit is around 1 MB (it is not advisable to submit codes of size close to that or larger). Once your code is compiled, the binary size should not exceed 1 MB.
- The compilation time limit is 30 seconds. You can find information about compilers that we use and compilation options here.
- There are 10 example test cases and 50 full submission (provisional) test cases.

Examples

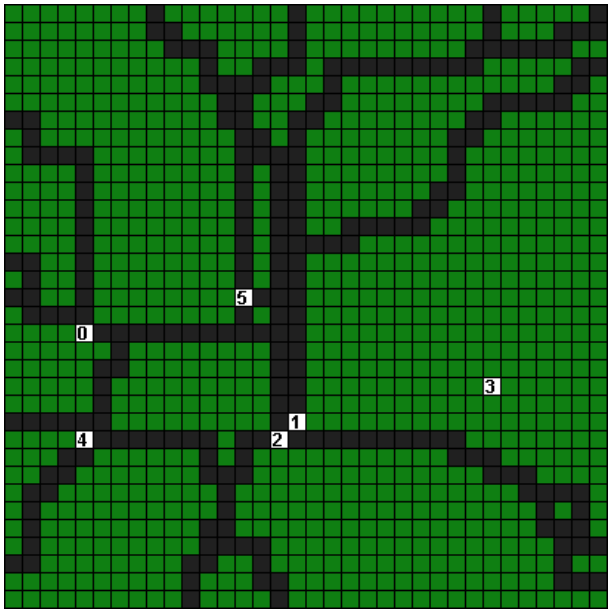
0)

```
seed = 1
```



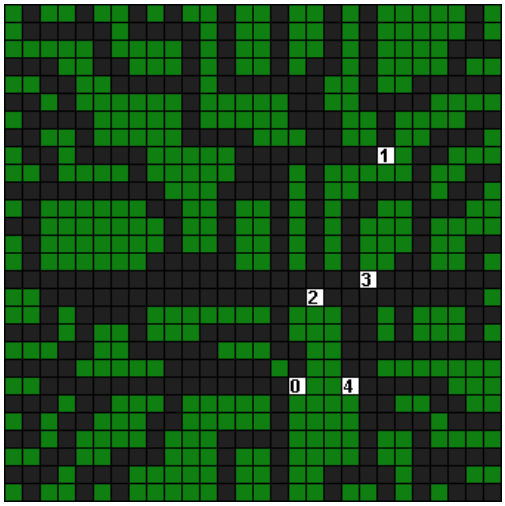
1)

```
seed = 2
```



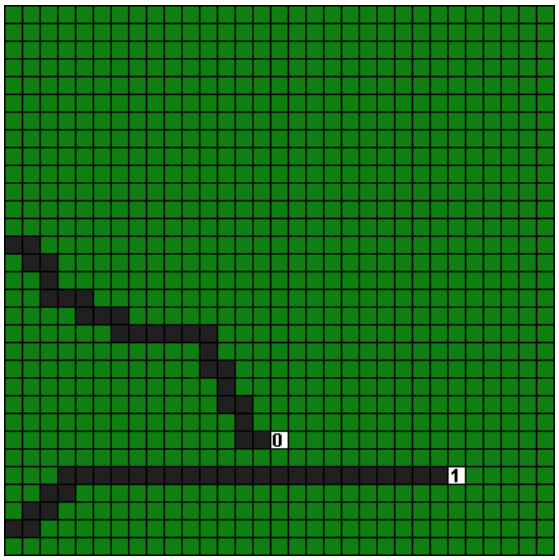
2)

```
seed = 3
```



3)

```
seed = 4
```



4)

```
seed = 5
```

5)

```
seed = 6
```

6)

```
seed = 7
```

7)

```
seed = 8
```

8)

```
seed = 9
```

9)

```
seed = 10
```
