Thankfully, we have done well in phase 1 in terms of grading, we wish to continue doing that well in phase 2, but to organize our work and save ourselves a significant amount of wasted time and to avoid the problems we have had in phase 1, I have written this document that I wish will be thoroughly read by all team members.

Rules to be followed when writing code:

The next few lines will setup some basic rules of the coding style to make sure that we share common ground and unify our editing characteristics to add more clarity and better cooperation.

I. Variable naming: Use the CamelCase convention when naming variables, google it if you're not familiar with the term.
II. Function naming: Use the CamelCase convention in case of nested words, but make sure to give the function a name that starts with an upper case letter if the name is a one word.
III. Use of braces: In a function, braces should start from the beginning on the new line "separate line form the one the prototype is written at", but at any other structure loops, conditionals, etc. the braces should start at the same line that header of the structure is written at.
IV. Spacing: Kindly, try to separate each code segment by the appropriate amount of spacing that you see suitable, but NEVER write the whole thing without any spacing, always consider readability.
V. Try to optimize the way you implement functions, if you have any doubt, seek help.
VI. Document your work using proper comprehensible English.
VII. Whenever you have created a function, declare it in the corresponding header and also don't forget to add a comment in the same style used in the header files to describe your functions, if further explanation is needed, feel free to add it as a comment above the function's body, refer to PrintEnemyByRegion( ) is documented for reference.

We, collectively will make sure that the above rules are being abided, if any team members have any kind of complaint on any of the above set of rules, feels free to express it, the same if you any further suggestions.

Functions from phase 1 that we will most probably keep using:

1- The I/O function.
2- The function Activate( ).
3- The function MoveFromTo( ).
4- The function DetachEnemy( ).
5- The function Kill( ).

The above function will be utilized in phase 2, maybe with some renaming or slight modifications to suit the new assigned tasks.

The logical view of the simulation:

1- Calls the function LoadFile( ) and extract all data from the input file, distribute it as per our logic to the corresponding data structures.

2- Initializes all needed variables, lists heads, defines queues, initializes counters, declares a castle, initializes the time step etc.

3- Calls the function Activate( ) on both inactive regular enemies along with inactive shielded enemies "using the same function we have used in phase 1".

4- Calls TowerShoot( )
The TowerShoot( ) function should perform the following the operations 4 times, one time for each tower:

- The tower first checks the active shielded enemies list and call the function CalculatePriority( ) which in turn will calculate the shielded enemy priority for each of the active shielded enemies. This value will be stored in a new struct attribute that we will add to the enemy struct.
- Calls the function PickToShoot( ) , which in turn will select N shielded fighters having the highest priority values, stores pointers to those selected enemies in an array of pointers that will be returned to the calling function.
- Calls the function TowerToEnemyDamage( ) and passes the array pointers obtained from the previous function, this function should do the following :
  - ❖ Uses the information of each enemy to calculate the damage done by the tower to this particular enemy.
  - ❖ Uses the values obtained from the calculations to reduce enemies' health, then checks whether the health reduction caused a zero health or not, if true, it should do the following :
    - ➢ Set KillTime to the current time step.
    - ➢ Calculate the Kill Delay (KD).
    - ➢ Calculate the Fight Time (FT).
    - ➢ Stores all the above values in new attributes that will be added to the enemy struct "We will thoroughly discuss this point".

5- Calls EnemyShoot( )

The function EnemyShoot( ) should perform the following steps as many time as the number of active enemies “Both regular and shielded, excluding pavers” :

- Calls EnemyToTowerDamage( ) that should do the following to fighters ONLY:
  - ❖ Traverses both enemy lists and for each enemy the function checks its region and based upon it, calculates the damage done by the enemy to this particular tower and accumulates that value in a variable named after the tower region, it should also mark the current enemy who made a shot as Reloading.
  - ❖ Stores those 4 accumulated values resembling that total damage done to each of the 4 towers and uses them to reduce each of the towers health.
- Calls the function CheckDestruction( ) that will check each of the 4 towers’ health, if any of them reached the value of zero or below “think about it, could happen”, it should perform the following steps :
  - ➢ Set a Boolean flag names destroyed as True “a new attribute that will be added to the tower struct”.
  - ➢ Calls the function RelocateEnemies( ) that will do the following :
    - For all inactive enemies “Both” change the current region value to the next region with a clockwise convention as described in the project description.
    - For all currently active enemies “Both” change the current region value to the next region with a clockwise convention as described in the project description, it should also simultaneously change the value of the DistanceToTower attribute as follows :
      - Check whether the current DistanceToTower is paved in the next tower or not, if true, leave it untouched.
      - If false, reset the value DistanceToTower to the currently paved distance of the tower to which the enemy will be relocated.

6- Calls Pave( ), this function do the following :

- Traverses the active regular enemies list, looks for pavers in a certain region, for each paver, it will uses the value FirePower “which in a paver’s case resembles their paving capacity” to reduce the UnpavedArea attribute in the tower struct, then marks that paver as Reloading.

7- Calls MoveEnemies( ), this function should do the following:
   - Traverses enemies lists “Both”, looks for all idle fighters “excluding pavers” in a certain region and prepare them for advancing one metre towards the tower only if the next metre is paved, in other words if UnpavedAread – DistanceToTower > 0 “we will discuss this point”, then the function will decrement the DistanceToTower by one metre, this should be repeated for all 4 regions.
   - Then we do the same as above with active pavers marked as reloading “we need to perform the same check as well”.

8- The following are some necessary steps that I still haven’t considered where to place them or what their order of appearance should be relative to other functions :
   - Calling DrawCastle( ).
   - Calling DrawRegions( ).
   - Calling DrawEnemies( ).
   - Printing Statistics, and if we decided to dedicate a function for that purpose.
   - Implementing the simulation mode details “silent, interactive, step-by-step”.
   - Any kind of deletion that involves the data we hold.
   - Organizing and preparing our data to pass them exactly the functions in the utility library expects to avoid having to go through the whole library editing stuff.
   - Reload period check, that should be performed to remove the mark from the reloading enemies whenever their reloading period terminates.

Finally, I just want to point out that the above logical view is just a preliminary view resulted while reading the project description, it might need editing, of course it will, but I expect it will be enough to finish 85% of the work left before any need to add more on to it.

Also, I want to point out the fact that I have intentionally left all the functions implementation details to the person whom the task will be assigned.

All tasks that we collectively distribute will have deadlines for delivering, and will go through testing, but whenever something is to be edited, all team members MUST be present or at least the TESTER and AUTHOR, unexpected modifications is NOT allowed for the sack of clarity.

I wish us the best of luck, isA :V

End of text.

Return 0 ;