Data Structure project

WAR

Description of the Game

What does this document contain?

This document gives you a precise explanation of how the cycle of a one time-step goes on, then a brief clarification of the most important functions, and finally some unique solutions of the problems we faced and their complexities.

Cycle of a regular one time-step

- When you press any of the three modes, which we will talk about their swaps later, you enter the “simulation” function, this function loops until the enemies are all killed or towers are, first, we loop on the “inActive” enemies to add the “Active” ones to the drawing list “BEnemiesForDraw” and to their towers, where each tower has a list of its active enemies.
- Six functions is called then, “setPriority” to update the priorities of the shielded, “MoveEnemies” to move each enemy, “AttackCastle” to make the enemies attack their towers, “AttackofTower” each tower attacks its enemies, “RemoveKilledEnemies” knowing that each tower removes the killed enemies from their “Active” lists, we have to remove them from the drawing function too, and “upDateEnemyTime” to update the attack time of each enemy.
- Then drawing the battle area (castle, paved area, enemies), and printing the information at that time-step.

Important Functions

- "void Castle :: MoveEnemies()" this function is responsible for looping on towers and making each one responsible for moving his own enemies by "Move()" function, the move function loops on all the enemies to move each one, with an exception for the paver when its attack time equals zero, the function decrements the distance of the enemy by the value of its speed, of course the enemy moves if the area is paved.
- "AddEnemy() & RemoveKilledEnemies()" these two functions is responsible for adding and removing enemies from a certain list "BEnemiesForDraw" which is responsible of drawing the enemies in the battle area.
- "void Castle::AttackCastle()" the enemies of each region attack the region's tower, We loop on all the towers and call function attack of the enemies then send to each enemy the tower wanted to be attacked, if the enemy in its attack time it will attack the tower by decrementing its health in case of 'shielded' or 'fighter', and decrementing the unpaved distance in case of 'paver', while 'supporter' raises enemies health to live longer, and finally 'tower controller' stops the towers from attacking the enemies, at the end of the attacking process we search for destroyed towers, if we found a destroyed tower we change the region of its enemies to the next **Alive** tower in the sequence of A->B->C->D and repeat.
- "void Tower::AttackEnemy(int Tstep)" each tower attack the enemies in its region if not banned by 'tower-controller' enemy...first we sort the shielded enemies then attack firstly the shielded enemies and if No. Of

shielded enemies less than the No. Of enemies the tower must shoot at each time step….the tower attack the rest of the enemies till reach the total number it can shoot at the time- step (N).

Unique Solutions and Functions

- One of the problems was swapping between modes (StepByStep, InterActive, Silent) so to get rid of it, we found that to get any action the user does during the game with ‘step by step’ mode we can't wait mouse click as it's not the logic of that mode so instead, we use “get mouse click” function, as if the user takes any action during “step by step” mode, we get this action and send it to simulation to change the current mode to the chosen mode by the user.
- The sort used here is merging sort, we had to sort the ‘Shielded’ enemies during the battle, to reset their priorities, to allow the towers to hit the enemy with highest priority.
- Using the function “getEnemy” this function is used to traverse both lists of the enemies….”Active” and “shielded-active” enemies...and send a pointer to the enemy by reference...it takes two parameters ‘bool’ taken by reference, refers to the beginning of the list shielded or active...and an integer value is equal to 1 if we traverse the active list and 2 if we traverse the shielded one.
- Lastly the reference pointer: To traverse the list without getting the head we make a temp pointer and if we want to traverse any list we set the reference to the head then advance the reference which make the deletion becomes O(1).