**Data Structure Project**

**Document**

**Castle vs Enemies**

**Team 17:**

Hady Maher Abd El-Samee

Mohamed Magdy

Mai Mahmoud

**Note**: All The Lists Are Linked Lists And There Is A Pointer To The First Enemy And Another To The Last.

We Made Two New Enemies:

**Sinper**: This Enemy Can Shoot The Tower From Far Distance And Can’t Approach The Castle (Min Distance = 45 m).

**Split**: This Enemy Can’t Shot The Tower But He Can Split To Be 8 Of Enemies And Has A High Priority To Prevent The Tower From Shooting The Other Enemies.

**Structures**

enum Etype {

PVR,

FITR,

SHLD\_FITR,

SNIPER,

SPLIT

};

struct enemy

{

int ID; //Each enemy has a unique ID (sequence number)

REGION Region; //Region of this enemy

int Distance; //Distance to the castle

double Health; //Enemy health

Etype Type; //PVR, FITR, SHLD\_FITR

int timeArrival; //Time Arrival Of Enemy

double power; //Enemy Power

int reloadPriod; //Enemy Reload Period

double Priority; // Priority for Enemy Shielded / Splitter

int NumberofSplits; //Number Of Splits Of The Splitter Enemy

int fightDelay; //Enemy Fight Delay

int killDelay; //Enemy Kill Delay

int RemainingTime; //Enemy Remaining Time To Shoot

int speed; //Enemy Speed

enemy \*next; //pointer To The Next Enemy

};

struct Tower

{

int TW; //Tower width

int TL; //Tower Height

double Health; //Tower Health

int N; //The Number Of Enemies That The Tower Can Attack In The Timestep

double towerPower; //Tower Shoot Power

int TPaved; //Paved distance of 60

int Numberofkilled; //Number Of Killed Enemies In The Tower Region

enemy \*active; //Pointer To The First Enemy In The Active Low Priority List

enemy \*activelast; //Pointer To The Last Enemy In The Active Low Priority List

enemy \*high; //Pointer To The First Enemy In The Active High Priority List

enemy \*highlast; //Pointer To The Last Enemy In The Active High Priority List

};

struct castle

{

int Xstrt;

int Ystrt;

int W; //width

int L; //Height

Tower towers[4]; //Castle has 4 towers

};

**Our Header Files**

1-Enemy motion.h

2-utility.h

3-Lists.h

4-listOperations.h

5-Fight.h

6-Simulation.h

**Our Source Files**

1-Main.cpp

2-Input-Output File.cpp

3-fight.cpp

4-listOperations.cpp

5-lists.cpp

6-utility.cpp

7-Simulation.cpp

8-Enemy motion.cpp

**1-** **Enemy motion.h:**

This Header File Contains Prototypes Of The Functions which Moves The Active Enemies In Source File **Enemy motion.cpp.**

* 1. **Enemy motion.cpp:**
     1. void DecreseDistance(castle &ct)

This Function Moves The Active Enemies By The Speed Of Them.

* + 1. void checkTowers(castle &ct, enemy\*&head)

This Function Check Check If Any Tower Is Dead call function ChangeRegion for both active and active high priority and inactive list

* + 1. void ChangeRegion(castle &ct, enemy\*temp) Check towers will call this function to move the enemies to next tower (**Clockwise**) -> of every enemy in active list or inactive

**2-utility.h:**

This Is The Header File In The Sample We Took.

We Made Some Changes In The Functions That Draw The Enemies, The Regions And The Towers.

**1-**The Function That Draw Enemies: It Draw Now Lists Not Arrays And Draw To Lists in One Time.

**2-**The Function That Draw Regions: It Draw the Unpaved Region with Deferent Color.

**3-**The Function That Draw Castle: It Now Draw The Towers In Red Color When It Damaged.

**3-Lists.h:**

This Header File Contains Functions In Source File **lists.cpp**:

1. void activeList(enemy \*&head, enemy \*&last, castle &ct, int timestep)

This Function Makes The Active Lists By The Time Arrival Of Each Enemy.

1. void enemy\_killed(castle &ct, int timestep, enemy \*&killedHead, enemy \*&killedLast)

This Function Makes The Killed List By Removing The Killed Enemies In From The Active Lists To The Killed List And Sort Them By The Killed Time And The Enemies Who Have The Same Killed Time Are Sorted By The Fight Delay.

**4-listOperations.h:**

This Header File Contains The Source File **listOperations.cpp** And Its Functions Make The Operations We Need In All Lists.

1. bool isEmpty(enemy \*head)

This Function Returns True If The List Is Empty And False Otherwise.

1. void insert(enemy \*&head, enemy \*&last, enemy \*data)

This Function Insert The Data In The Last Of The List By Taking The Enemy Data In A Pointer And The Two Pointer (**head**) The Pointer To The First Enemy In The List And (**last**) The Pointer To the Last Enemy In The List And Insert The Data In The Last Of The List.

1. int countList(enemy \*head)

This Function Returns Number Of Enemy In The List.

1. void destroy(enemy \*&head, enemy \*&last)

This Function Destroy All The Enemies In The List By deleting All Enemies One By One And Make The (**head & last**) Equal NULL.

1. int countList\_byregion(enemy\*head, int x)

This Function Returns Number Of The Enemies in A Specific Region.

1. void remove\_from\_beg(enemy \*&head, enemy \*&last)

This Function Remove The First Enemy In The List.

1. void MergeSorted(enemy \*&head, enemy \*&last, enemy \*&appended, enemy \*&appendedlast)

This Function Merge To Sorted Lists (Sorted By Time Arrival).

**5-Fight.h:**

This Header File Contains Functions In The Source File **fight.cpp** And Its Functions Do The Fight Logic.

1. void checkperiority(enemy\*&head, enemy\*&last, double C1, double C2, double C3,int N)

This Function calculate The Priority Of The High Priority Enemies According To The Parameters (**C1 & C2 & C3**).

Then Sort The List According To The Priority.

1. void DecreaseHealth\_Enemy(enemy \*&shooted, double TP)

This Function Decrease The Shooted Enemy Health According To The Given Equation.

1. void DecreaseHealth\_Tower(Tower &shooted\_tower, enemy \*shooter)

This Function Decrease The Shooted Tower Health According To The Given Equation.

1. void Fight\_Tower\_Enemy(castle &ct, double C1, double C2, double C3, enemy \*&killedHead, enemy \*&killedLast, int TimeStep)

This Function Make The Tower Shoot The Enemies And Call The Functions That Calculates The Priority And The Closer Enemy …etc.

It Makes The Fight Logic Between The Tower And The Enemies.

1. void Fight\_Enemy\_Tower(castle &ct)

This Function Make The Enemies Attack The Towers.

It Makes The Fight Logic Between The Enemies And The Tower.

1. void generate\_splitter(castle &ct , enemy\*e)

This Function Generates The Splitter Enemy

**6-Simulation.h:**

This Header File Contains The Functions In Two Source Files **Simulation.cpp** And **Input-Output File.cpp.**

These Functions read The input From The Input File And Simulates The Game And Make The Output File.

1. **Simulation.cpp:**
   1. bool GameOver(castle ct, enemy\*head)

This Function Returns False If The Tower Or The Enemies Are All Killed.

* 1. void Simulation(enemy \*&head, enemy \*&last, int TH , int N , int TP, double C1 , double C2 , double C3)

This Function Reads The Data From The Input File, Simulates The Game And Make The Active List , Move The Enemies , Kill Them, Make The Output File And Call Almost All The Functions In The Program …etc.

**2- Input-Output File.cpp:**

1. void inputfromfile(enemy \*&head, enemy \*&last, double &c1, double &c2, double &c3, int &TH, int &N, int &TP)

This Function Takes The Input From The Input File (input.txt) And Return The Data In The inactive List And The Tower Properties and Priority Parameters.

**Parameters:** A pointer **(head)** To The First Enemy In The Inactive List And A Pointer (**last**) To The Last Enemy In It.

(**c1 & c2 & c3**) Priority Parameters.

(**TH**) Tower Health.

(**N**) The Number Of The Enemy That The Tower Can Attack in The Time Step.

(**TP**) Tower Power.

1. void Out(castle ct, enemy \*Killed, enemy \*head, double TH)

This Function Makes The Output File.

**Parameters**: The Castle (**ct**)

A pointer (**Killed**) To The First Enemy In The Killed List

A Pointer (**head**) To The First Enemy In The Inactive List

(**TH**) The Main Tower Health [To Calculate The Damage That The Tower Had].

1. double Avarage (enemy \*head, char ch)

This Function Returns The Average Of The Kill Delay And Fight Delay According To The Character (**ch**).

**7-Main.cpp:**

This Source File Contains **Main** Function That Chooses The Mode Of The Game And Call The Function (**Simulation**) That Make The Simulation Of The Program.