

Data-Structure Project

Semester 16' - Team #15

Tuesday, December 20, 2016

- Simulator Structure:

The project is separated into 3 main modules: Control, Data and Graph, each has its header and source files

Header files contain prototypes and source files contain the implementation

Each module contains different namespaces to represent submodules with their functions. e.g ENEMY, TOWER, Control etc.. The use of namespaces is to be able to use multiple functions with the same name for different modules, this approach does increase readability, maintenance and modularity of the project

NOTE: enemies have different speeds



screenshot of the simulator showing the Tank and two enemies

- To Increase Performance:

1. While looping in enemies, function ENEMY::Loop stops when it finds inactive enemy.
2. Function Control::Read() that loads data to memory, doesn't traverse the whole list of enemies to add one, it saves the last enemy assess in a pointer.
3. DrawEnemies() is overloaded to be able to take the whole castle instead of array of enemies, in order to avoid allocating special array for each region.
4. To Sort shielded enemies depending of priority, we stored their addresses ,instead of storing all of their data, in array, and sorted the addresses.
5. When passing big data to functions, we used constant reference to avoid passing huge amounts of value or the addresses themselves.

- Main Modules:

Control:

Contains main functions that control the flow of the simulator. e.g loop, start, get mood and refresh screen etc...

NOTE: some overwritten graph functions are in Control not in Graph

Data:

Contains data structures: Enemy, Castle and Tower, each contains its attributes (variables).

Also it has namespaces for each data module, each one has functions/ helper-functions for that module

Graph:

Contains provided graphing (drawing) functions. It is a rename of utility

We added some functionality in it to support drawing Tank enemy

Overloaded functions of DrawEnemies are written in Control to avoid changing the original.

- Data Structure:

We avoided using OOP, instead we made structs to hold variables and exported its functionality in namespaces
We didn't modify given variables in structs, but added new
Here provided screenshots of the modifications:

Enemy

```
193 struct Enemy
194 {
195     // Given Properties :-
196     int ID;           //Each enemy has a unique ID (sequence number)
197     REGION Region;    //Region of this enemy
198     int Distance;     //Distance to the castle, initialized to 0 (Inactive)
199     int Health;       //Enemy health
200     TYPE Type;        //PVR, FITR, SHLD_FITR
201
202     // Modified Properties :-
203
204     // from input:
205     int arrive_time;
206     int fire_power;    // if paver, it is num of metres it can pave
207     int reload_period;
208     int speed;         // speed of object (bonus)
209
210     // to be calculated for output
211     int fight_delay;   // time of begin fighting - time of arrival
212     int kill_delay;    // time of kill - time of arrival
213     // initialized to -1, to catch the bug if it is still -1 at end of program
214
215     double priority;   // initialized to -1
216     // Pointers
217     Enemy* next;       // initialize to NULL
218     Enemy* prev;       // initialize to NULL
219
220 };
```

ENEMY NAMESPACE:

```
80 namespace ENEMY
81 {
82     Enemy* Initialize(const int &S, const int &TY, const
83     void Loop(Enemy* e, Tower* T, const int &timer, En
84     Enemy* Add(Tower* t, Enemy* &lastOne,
85         const int &S, const int &TY, const int &T, cons
86         const int &Pow, const int &Prd, const int &Spee
87     void Move(Enemy &e, const Tower &T);
88     Enemy* AddToDead(Enemy* e);
89     void Print(const Enemy &e);
90     void Swap(Enemy* &a, Enemy* &b);
91     bool IsActive(const Enemy &e, const int &time);
92     void Kill(Enemy* e, Tower* t, const int &time);
93     bool CanFire(Enemy* e, int time);
```

Tower:

```
162 struct Tower
163 {
164     // Given Properties :-
165     int TW; //Tower width
166     int TL; //Tower Height
167     int Health;
168
169     // Modified Properties :-
170
171     // from input:
172     int unpaved;          // the pos of first unpaved block, initialized to 30
173     int fire_power;
174     int maxN_enemies;     // max num of enemies it can fight
175
176     // Pointers to enemies list
177     Enemy* firstEnemy;    // initailized to NULL
178     Enemy* firstShielded; // initialized to NULL
179     int num_enemies;      // increamented by one when adding enemy, referes to all enemies
180 };
---
```

TOWER NAMESPACE:

```
62 namespace TOWER
63 {
64     extern double c1, c2, c3;          // constants read in run-time to calculate priority enemies
65
66     void Initialize(Castle &c, const int &TH, const int &N, const int &TP);
67     bool IsEmpty(const Tower &t);
68     void Fire(Tower* t, Enemy* arr[], int size, int time);
69     bool IsDestroyed(const Tower &t);
70     void Damage(Enemy* e, Tower* t);
71     void Transfer(Castle &c, int region);
72     void _Transfer(Tower* T1, Tower* T2, TYPE type, const REGION &Region);
73     void _Fix_forTransfer(Tower* T, Enemy *e, const REGION &Region);
74     void Destroy(Tower* T);
75     bool HasFinished(Tower &T);
76     int GetNumOfShielded(Tower *T);
77     int GetNumOfNormal(Tower *T);
78 }
```

Castle:

CASTLE NAMESPACE:

```
51 namespace CASTLE
52 {
53     void Initialize(Castle &C);
54     void Loop(Castle &c, const int &timer);
55     bool IsEmpty(const Castle &c);
56     bool IsDestroyed(const Castle &c);
57     int GetTotalEnemies(const Castle &c);
58     void Destroy(Castle &c);
59     bool HasFinished(Castle &c);
60 }
```

other:

Project included other modules like:

- **Log** (handle output file data and printing statistics to screen)

```
121 namespace Log
122 {
123     // bunch of variables to store information to be print at end
124     extern int total_FD;
125     extern int total_KD;
126     extern int total_enemies_beg;           // at beginning
127     extern int tower_health_beg;           // at beginning
128     extern int last_killed[NUM_OF_TOWERS];
129     extern int all_killed[NUM_OF_TOWERS];
130
131     // to init the file
132     void Initialize(Castle &c);
133
134     // add enemy to file
135     void ToFile(Enemy* e, const int &time);
136
137     // add towers data to file
138     void ToFile(const Castle &c);
139
140     // end of file, state is the state of the game
141     void End(const Castle &c);
142
143     void ToScreen(const Castle &c);
144 }
```

- **Shielded** (contains functions to handle priority of Shielded enemies)
- **Doctor** (contains Heal() function that increases health of two (if available) enemies around him)
- **Paver**
- **Tank**

- **New Enemies:**

We added 2 new enemies: Doctor and Tank, they only take action when their fire period finish

- *Doctor:*

Heals the two enemies near to him, each enemy's health is increased by doctor fight power / 2
Doctor can't heal a tank

colour is WHITE

- *Tank:*

Generates new random enemies to the list behind him, each one has arrival time = Tank arrival time + 1
Tank doesn't generate Tanks

NOTE: tanks leads sometimes to unpredicted behaviour, you cant predict exactly how will be the results.

Shape is 8 grey # with * in the middle