Класифікація сутностей гри-стрілялки

Виявлені сутності:

Гармата

має: координати, розміри, вигляд

вміє: зображати себе, рухатись ліворуч-праворуч, стріляти – породжувати кулю

Куля

має: координати, розміри, вигляд

<u>вміє</u>: зображати себе, рухатись догори, вилітати за межі екрана, повідомляти свої координати, вибухати (?)

Літак

має: координати, розміри, вигляд

вміє: зображати себе, рухатись праворуч, вилітати за межі екрана, перевіряти влучання, вибухати

Аеродром

пам'ятає: інтервал часу до надсилання наступного літака

<u>вміє:</u> опрацьовувати повідомлення від таймера, генерувати літак з випадковими координатами та швидкістю

Список куль

містить: однозв'язний список куль

<u>вміє:</u> додавати нову кулю, вилучати кулю (що втекла чи вибухнула), зображати кулі, рухати кулі, знаходити кулі, що вилетіли за межі екрана, перебирати кулі, щоб виявити влучання

Список літаків

містить: однозв'язний список літаків

<u>вміє:</u> додавати новий літак, вилучати літак (що втік чи вибухнув), зображати літаки, рухати літаки, знаходити літаки, що вилетіли за межі екрана, перебирати літаки, щоб виявити влучання

Клавіатура

вміє: розпізнавати натиснуті клавіші, надсилати повідомлення гарматі

Таймер

<u>вміє:</u> відслідковувати задані інтервали часу, надсилати повідомлення аеродромові, спискові куль і спискові літаків

Арена

пам'ятає: розміри ігрового поля, тло

вміє: повідомляти розміри, зображати тло

Рефері

містить: лічильники пострілів, збитих і пропущених літаків

вміє: обчислювати рахунок гри

Споріднені сутності та ієрархії класів

Рухомі об'єкти

Гармата, куля, літак мають однакову структуру і схожу поведінку (вміють зображати себе). Куля і літак ще більше схожі, бо можуть вилетіти за межі екрана, вибухнути. Кожна сутність має свої особливості: гармата – стріляє, куля – повідомляє свої координати, літак – перевіряє влучання.

```
// Абстрактний рухомий об'єкт. Містить всі
                                                   дані, необхідні нащадкам
            TMovedObject
            static HBITMAP GunView, ...,
                                              // Змінні класу містять растрові зображення,
                   AircraftMask;
                                                   потрібні для відображення об'єктів
            static void InitBitmaps(
                                              // Метод класу завантажує растрові
                   HINSTANCE);
                                                   зображення з ресурсів аплікації
            int x, y;
                                              // Координати лівого верхнього кута
            int width, height;
                                              // Розміри
            int step;
                                              // Швидкість переміщення
            HBITMAP view, mask;
                                              // Растрове зображення об'єкта
            HDC hMemoryDC;
                                              // Контекст пристрою для виведення зображ.
            void show(CpaintDC&);
                                              // Незалежно від типу всі об'єкти
                                                   відображають однаково
TAutoMovedObject: TMovedObject
                                            TGun: TMovedObject
// тип об'єктів, що літають самі
                                            void moveLeft();
bool isOut() = 0;
                                            void moveRight();
void move() = 0;
                                            TBullet* fire();
TBullet: TAutoMovedObject
                                  TAircraft: TAutoMovedObject
bool isOut():
                                  int killed;
void move();
                                  bool isOut();
int getX();
                                  void move();
int getY();
                                  void crash();
                                  bool hitBy(const TBullet&);
```

У ієрархії ϵ проміжний абстрактний клас, що об'єдну ϵ типи об'єктів, які рухаються самі. Він ϵ основою для створення колекцій рухомих об'єктів.

Літак може перебувати у двох станах: звичайний літак рухається горизонтально, підбитий літак змінює зовнішній вигляд і рухається інакше, втрачаючи висоту. Через деякий час підбитий літак зникає. Поле killed — поле стану об'єкта. У звичайного літака killed = 0 (false). Метод crash() змінює стан об'єкта: $killed \ge 1$ (true), міняється зовнішній вигляд (поля view, mask), інакше починають працювати методи move(), isOut().

Базовий клас *TMovedObject* суттєво залежить від можливостей графічного середовища: він завантажує растри для зображення об'єктів, створює контексти пристрою для виведення растрових зображень, застосовує API-функції тощо. Щоб ізолювати його від цих особливостей і зробити мобільнішим, можна оголосити клас *TGraphicScreen*, що інкапсулює виведення графіки, та організувати взаємодію *TMovedObject* з *TGraphicScreen* замість безпосередньої взаємодії з ОС.

Контейнери об'єктів

Список куль і список літаків мало чим відрізняються. Все обслуговування зв'язної структури, призначеної для зберігання рухомих об'єктів можна винести в спільний базовий клас. Підкласи спеціалізуватимуться на своїх особливих способах перевірки влучання: список куль передає свої кулі спискові літаків, список літаків передає кулю кожному літакові для порівняння координат.

Базовий клас — лінійний однозв'язний список з заголовною ланкою. Заголовна ланка потрібна для того, щоб перша ланка списку не відрізнялася від усіх інших: її адресу, як і всіх інших, міститиме поле next попередньої (заголовної) ланки. Елементом ланки ϵ поліморфний вказівник на базовий клас рухомого об'єкта.

Додавати ланку можна в довільному місці списку, адже вилучення об'єктів зі списку непередбачуване, тому нема сенсу підтримувати якийсь порядок. Найлегше додавати на початок, одразу після заголовної ланки.

Рухомі об'єкти зображатимемо в графічному середовищі ОС Windows за допомогою класів бібліотеки MFC, тому список відображатиме свої елементи на контекст графічного пристрою.

```
// Список рухомих об'єктів. Більшість
TList
                                  // його дій не залежить від типу об'єктів
   Node
                                  // Ланка однозв'язного списку
   TAutoMovedObject* obj;
   Node*
                       next;
                                  // Список зберігає адресу першої ланки
Node*
             head:
                                  // Конструктор створює заголовну ланку
TList();
                                  // Деструктор знищує весь ланцюжок
~TList();
                                       разом з усіма об'єктами
                                  // Додавати можна завжди на початок
void put(TAutoMovedObject*);
                                  // Щоб вилучити ланку, потрібен доступ до
void remove(Node* pre);
                                       попередньої (до поля next)
void garbage();
void moveAll();
                                  // Об'єкти зображаємо на контексті
void showAll(CpaintDC&);
                                   // графічного пристрою
```

```
TBulletList: TList
void lookForHitIn(TAircraftList&);
// передає кожну свою кулю
// спискові літаків для порівняння
```

```
TAircraftList : TList
bool lookForShotedBy(TBullet&);
// порівнює кулю з кожним зі своїх
// літаків щоб виявити влучання
```

«Одинокі» об'єкти

Аеродром – простий клас, що отримує повідомлення від таймера, генерує нові літаки і додає їх до списку літаків. Може містити посилання на список літаків.

Клавіатура — не обов'язковий клас, бо натискання клавіш розпізнає операційна система. Достатньо передбачити опрацювання відповідного повідомлення. У ОС Windows це — WM_KEYDOWN.

Таймер – також не обов'язковий клас. Всі таймери – об'єкти операційної системи, доступ до яких забезпечує стандартний набір АРІ-функцій. Обгортання такого системного об'єкта власним класом може переслідувати лише одну мету – надати користувачеві (програмістові) зручніший або звичніший інтерфейс взаємодії.

Рефері – бажана, але не обов'язкова сутність. Все працюватиме і без неї.

Арена у найпростішому випадку – просто вікно діалогу МГС, автоматично згенерований клас.

```
// Клас головного вікна аплікації міститиме всіх
CGameDlg: CDialog
                                             учасників гри, стандартну функціональність
TAircraftList aircraftList;
                                             діалогу та специфічні для гри методи
             bulletList;
TBulletList
TGun
             Gun:
TAirfield
             airfield;
             hMemDC;
HDC
BOOL OnInitDialog();
                                        // допомагає конструкторові виконати початкові
                                             налаштування екземпляра діалогу
void OnPaint();
                                        // працює кожного разу, коли перемальовується
                                             вікно (виклик - Invalidate())
void OnKeyDown(UINT,UINT,UINT);
                                        // опрацьовує натискання клавіш
void OnTimer(UINT_PTR);
                                        // опрацьовує повідомлення системного таймера
void OnLButtonDown(UINT,CPoint);
                                        // опрацьовує клік лівою клавішею миші
```

Схема обміну повідомленнями

