Pràctica 3: Batalla de portaavions

Programació 2

Curs 2024-2025

Aquesta pràctica consisteix a simular una batalla entre caces de portaavions en una zona de conflicte, seguint el paradigma de programació orientada a objectes (POO). Els conceptes necessaris per desenvolupar aquesta pràctica es treballen en tots els temes de teoria, especialment al *Tema 5*.

Condicions de lliurament

- La data límit de lliurament d'aquesta pràctica és el **divendres 9 de maig**, fins a les **23:59**
- La pràctica consta de diversos fitxers: `Coordinate.cc`, `Coordinate.h`, `Fighter.cc`, `Fighter.h`, `AircraftCarrier.cc`, `AircraftCarrier.h` `Board.cc` i `Board.h`. Tots ells s'hauran de comprimir en un únic fitxer anomenat `prog2p3.tgz` que s'entregarà a través del servidor de pràctiques de la forma habitual. Per crear el fitxer comprimit ho has de fer de la manera següent (en una única línia):

Terminal

```
$ tar cvfz prog2p3.tgz Coordinate.cc Coordinate.h Fighter.cc Fighter.h
          AircraftCarrier.cc AircraftCarrier.h Board.cc Board.h
```

Codi d'honor

Si es detecta còpia (total o parcial) a la teua pràctica, tindràs un **0** en el lliurament i s'informarà a la direcció de l'EPS perquè adopte mesures disciplinàries

És correcte discutir amb els teus companys possibles solucions a les pràctiques
És correcte apuntar-te a una acadèmia si serveix per obligar-te a estudiar i fer pràctiques

És incorrecte copiar codi d'altres companys o demanar a ChatGPT que et faça la pràctica
És incorrecte apuntar-te a una acadèmia perquè et facen les pràctiques

Si necessites ajuda, dirigeix-te al teu professor/a
No copies

Normes generals

- Has de lliurar la pràctica exclusivament a través del servidor de pràctiques del Departament de Llenguatges i Sistemes Informàtics (DLSI). S'hi pot accedir de dues maneres:
  - Pàgina principal del DLSI (`https://www.dlsi.ua.es`), opció "LLIURAMENT DE PRÀCTIQUES"

    - Directament a l'adreça https://pracdlsi.dlsi.ua.es
- Qüestions que has de tenir en compte en fer el lliurament:
    - L'usuari i la contrasenya per lliurar pràctiques són els mateixos que utilitzes a UACloud
    - Pots lliurar la pràctica diverses vegades, però només es corregirà l'últim lliurament
    - No s'acceptaran lliuraments per altres mitjans, com el correu electrònic o UACloud
    - No s'acceptaran lliuraments fora de termini
- La teua pràctica ha de poder ser compilada sense errors amb el compilador de C++ existent a la distribució de Linux dels laboratoris de pràctiques
- Si la teua pràctica no es pot compilar, la seua qualificació serà 0
- Al començament de tots els fitxers font lliurats has d'incloure un comentari amb el teu NIF (o equivalent) i el teu nom. Per exemple:

Coordinate.h

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- Superar **totes** les proves de l'autocorrector et dóna dret a presentar-te a l'examen de pràctiques. Si falla alguna prova no podràs presentar-te a l'examen i la teua qualificació en aquesta pràctica serà 0
- El càlcul de la nota de la pràctica i la seua rellevància en la nota final de l'assignatura es detallen a les transparències de presentació de l'assignatura (*Tema 0*)

1. Descripció de la pràctica

En aquesta pràctica s'implementaran diverses classes que permetran simular una batalla entre dos portaavions, sobre un tauler que representa la zona del conflicte. Els portaavions enviaran els seus caces a lluitar per cada posició del tauler.

2. Detalls d'implementació

Al *Moodle* de l'assignatura es publicaran diversos fitxers que necessitaràs per a la correcta realització de la pràctica:

- `prac3.cc`. Fitxer que conté el `main` de la pràctica. S'encarrega de crear els objectes de les classes implicades en el problema i simular una batalla. Aquest fitxer no s'ha d'incloure en el lliurament final, és només una ajuda per provar la pràctica, i es pot modificar o fer servir com a base per a altres proves
- `makefile`. Fitxer que permet compilar de manera òptima tots els fitxers font de la pràctica i generar un únic executable
- `autocorrector-prac3.tgz`. Conté els fitxers de l'autocorrector per provar la pràctica amb diverses proves unitàries per provar els mètodes per separat. La correcció automàtica de la pràctica, després del lliurament, es realitzarà amb aquestes mateixes proves i serà necessari superar-les totes per poder presentar-se a l'examen d'aquesta pràctica.

En aquesta pràctica cadascuna de les classes s'implementarà en un mòdul diferent, de manera que tindrem dos fitxers per a cadascuna d'elles: `Coordinate.h` i `Coordinate.cc` per a les coordenades en el tauler, `Fighter.h` i `Fighter.cc` per gestionar els caces, `AircraftCarrier.h` i `AircraftCarrier.cc` per als portaavions, i `Board.h` i `Board.cc` per al tauler. Aquests fitxers, juntament amb `prac3.cc`, s'han de compilar conjuntament per generar un únic executable. Una manera de fer-ho és de la manera següent:

Terminal

```
$ g++ Coordinate.cc Fighter.cc AircraftCarrier.cc Board.cc prac3.cc -o prac3
```

Aquesta solució no és òptima, ja que compila de nou tot el codi font quan pot ser que només algun dels fitxers haja sigut modificat. Una manera més eficient de realitzar la compilació de codi distribuït en diversos fitxers font és mitjançant l'eina `make`. Has de copiar el fitxer `makefile` proporcionat al Moodle dins del directori on estiguen els fitxers font i introduir la següent ordre:

Terminal

```
$ make
```

- Pots consultar la transparència 61 en endavant del *Tema 5* si necessites més informació sobre el funcionament de `make`

2.1. Excepcions

Alguns dels mètodes que crearàs en aquesta pràctica han de llançar excepcions, que són un mecanisme especialment útil en els constructors perquè eviten construir objectes amb valors incorrectes.

Per llançar una excepció has d'utilitzar `throw` seguit del tipus d'excepció, que en C++ pot ser qualsevol valor (un enter, una cadena, etc), però en aquesta pràctica ha de ser sempre l'excepció estàndard `invalid_argument` (que mai no s'ha de capturar):

Terminal

```
if (...)
 throw invalid_argument(missatge);
```

on `missatge` és una cadena (`string` o vector de `char`); si l'argument invàlid és un nombre, es pot convertir a cadena amb la funció `to_string`:

Terminal

```
if (...)
 throw invalid_argument(to_string(nombre));
```

Per fer servir `invalid_argument` has d'afegir al teu codi `#include <stdexcept>`

2.2. Generació de nombres aleatoris

Per a la simulació de la lluita entre dos caces és necessari generar un nombre aleatori, utilitzant aquesta funció (que has d'incloure al principi del fitxer `Fighter.cc`):

Terminal

```
#include <cstdlib> // Per a rand()

// retorna un nombre aleatori entre 0 i maxrnd-1
int getRandomNumber(unsigned int maxrnd){
  return rand() % maxrnd;
}
```

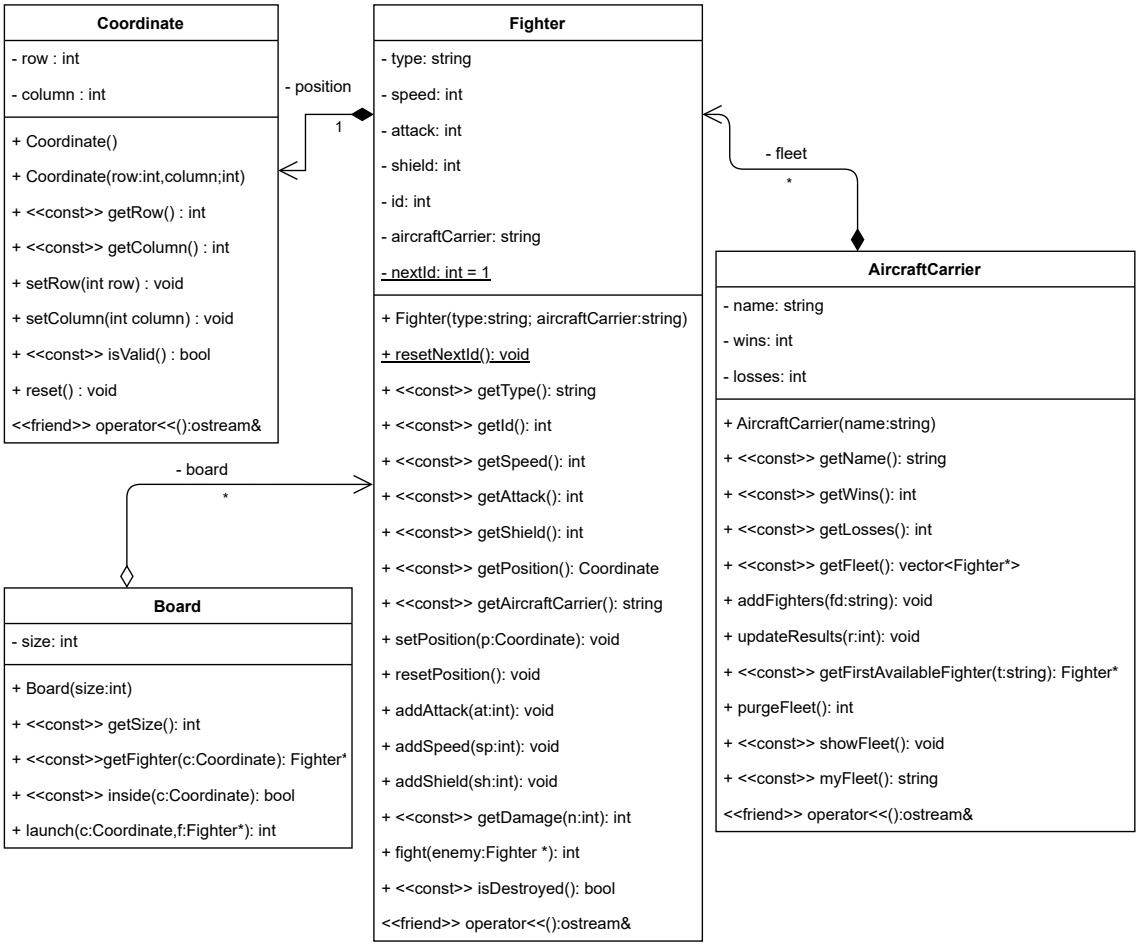
- Al fitxer `prac3.cc` s'inicialitza el generador de nombres aleatoris cridant a la funció `srand`. No has de modificar aquesta crida.

3. Classes i mètodes

La figura que apareix a la pàgina següent mostra un diagrama UML amb les classes que cal implementar, juntament amb els atributs, mètodes i relacions que tenen lloc a l'escenari d'aquesta pràctica.

En aquesta pràctica no està permès afegir cap atribut ni mètode públic a les classes definides en el diagrama, ni afegir ni canviar arguments dels mètodes. Si necessites incorporar més mètodes i atributs a les classes descrites en el diagrama, pots fer-ho, però sempre incloent-los a la part privada de les classes. Recorda també que les relacions d'*agregació* i *composició* donen lloc a nous atributs quan es tradueixen del diagrama UML a codi. Consulta les transparències 58 i 59 del *Tema 5* si tens dubtes sobre com traduir les relacions d'*agregació* i *composició* a codi.

A continuació es descriuen els mètodes de cada classe. És possible que alguns d'aquests mètodes no siguen necessaris per a la pràctica, però s'utilitzaran en les proves unitàries durant la correcció. Es recomana implementar les classes en l'ordre en què apareixen en aquest enunciat.



3.1. Coordinate

Aquesta classe gestiona les coordenades en el tauler. Com es pot veure en el diagrama, té dues variables d'instància privades, "`row`" i "`column`", que representen la fila i la columna en el tauler, respectivament. Els mètodes d'aquesta classe són els següents:

- `Coordinate()`. Crea una coordenada assignant -1 a la fila i la columna. Aquesta coordenada seria una coordenada no vàlida
- `Coordinate(int row,int column)`. Crea una coordenada assignant el valor de `row` a la fila i `column` a la columna
- `int getRow() const`. *Getter* que retorna la fila
- `int getColumn() const`. *Getter* que retorna la columna
- `void setRow(int row)`. *Setter* que modifica la fila
- `void setColumn(int column)`. *Setter* que modifica la columna
- `bool isValid() const`. Retorna `true` si la fila i la columna són més grans o iguals que 0, i `false` en cas contrari
- `void reset()`. Assigna el valor -1 a tots dos atributs, com si fos una coordenada acabada de crear amb el constructor sense arguments
- `ostream& operator<<(ostream &os,const Coordinate &c)`. Operador d'eixida que mostra la coordenada. Per exemple, si la fila és 2 i la columna 7 mostrarà `[2,7]`. Si la coordenada no és vàlida mostrarà `[-,-]`. No s'ha d'afegir un salt de línia després d'imprimir aquesta informació per pantalla

3.2. Fighter

Aquesta classe reflecteix el comportament dels caces, i té diversos atributs (tots ells privats):

- `type`: una cadena (`string`) que indica el tipus de caça: `F-35B`, `EuroFighter`, ...
- `speed`: velocitat del caça (no ha de ser negativa)
- `attack`: capacitat d'atac del caça (no ha de ser negativa)
- `shield`: escut del caça; si arriba a ser 0 o negatiu es considerarà que el caça està destruït
- `id`: identificador, únic per a cada caça. Quan es crea un nou caça, se li assigna com a identificador el valor de `nextId` i s'incrementa `nextId`; com es pot veure al diagrama, inicialment `nextId` val 1
- `position`: coordenada que representa la posició del caça al tauler; inicialment serà una coordenada no vàlida (la fila i la columna valdran -1), i se li assignarà un valor quan es col·loque el caça al tauler; quan el caça siga destruït i s'esborre del tauler, ha de tornar a ser una coordenada no vàlida
- `aircraftCarrier`: nom del portaavions al qual pertany el caça

Al diagrama UML apareixen els mètodes d'aquesta classe; molts d'ells són *getters* simples, que retornen els camps als quals fan referència. Per exemple, `getType` retorna l'atribut `type` de l'objecte. Els mètodes `addAttack`, `addSpeed` i `addShield` actuen de manera similar als *setters*, però sumant el valor rebut com a argument a l'atribut (com indica el nom del mètode). A més, en cas que la velocitat (`speed`) o la capacitat d'atac (`attack`) resulten negatives després de la suma (l'argument podria ser negatiu), s'han de posar a `0`.

- `Fighter(string type, string aircraftCarrier)`. Constructor que inicialitza un caça, assignant un valor de 100 a la velocitat, 80 a l'atac i 80 a l'escut. A més, assigna els paràmetres als atributs `type` i `aircraftCarrier`, i assigna valor a l'identificador (i s'incrementa `nextId`). La coordenada s'inicialitzarà automàticament al valor per defecte (`-1,-1`), no cal fer res més. Si `type` és una cadena buida s'ha de llençar l'excepció `invalid_argument` amb la cadena `"Wrong type"`

- `void resetNextId()`. Mètode estàtic que inicialitza el valor de l'atribut estàtic `nextId` a 1. No s'ha d'utilitzar en la pràctica, però per realitzar proves unitàries és necessari
- `void resetPosition()`. Crida el mètode `reset` per a la posició del caça
- `int getDamage(int n) const`. Retorna el dany infligit pel caça al caça enemic, que serà sempre `(n*attack)/300`. Com que tots els números són enters, la divisió ha de ser entera i el resultat també
- `int fight(Fighter *enemy)`. Aquest mètode simula la lluita entre dos caces, el que rep la crida (el *caça*) i un caça enemic (`enemy`, el caça *enemic*). Abans de començar, si algun dels dos caces ha estat destruït, retorna 0; en cas contrari, inicia un bucle que acabarà quan un dels dos caces resulte destruït (és una lluita a mort). En cada pas del bucle un dels caces atacarà l'altre; per fer-ho s'obtindrà un nombre aleatori `n` entre 0 i 99 (cridant a la funció `getRandomNumber`), i es compararà aquest valor amb un llindar u, que es calcula amb la següent fórmula:

  $$u = \frac{100v_1}{v_1 + v_2}$$

  on v_1 és la velocitat del *caça*, i v_2 la de l'*enemic*. Com que tots els valors són enters, la divisió serà entera (no faces servir nombres reals).

  Si el llindar u és més gran o igual que el nombre aleatori `n`,[1] l' atacant serà el *caça* i es restarà a l'escut de l'*enemic* el dany causat pel caça (que serà el valor retornat pel mètode `getDamage` passant `n` com a paràmetre); en cas contrari, l'atacant serà l'*enemic* i es restarà a l'escut del *caça* el dany causat per l'*enemic* (cridant també a `getDamage` de l'enemic, però en aquest cas passant `100-n` com a paràmetre).

  Si després d'un atac un caça resulta destruït, la lluita acabarà i el mètode retornarà 1 si ha guanyat el *caça* o -1 si ha guanyat l'*enemic*; si cap dels dos ha estat destruït, es continuarà dins del bucle i es tornarà a obtenir un altre nombre aleatori.

  Per exemple, si $u = 50$ (ambdós caces tenen la mateixa velocitat) i el primer nombre aleatori n és 80, es restarà 5 a l'escut del caça (`this`); si el següent nombre és 88, es restaran 3 novament al caça, i si el següent és 13 es restaran 3 a l'escut del caça enemic (`enemy`). El bucle continuarà fins que un dels caces resulte destruït.

  IMPORTANT: assegura't que no crides `getRandomNumber` més d'una vegada dins del bucle, si fas més crides el resultat d'aquesta lluita (i les següents) probablement serà diferent i la teva pràctica funcionarà malament.
- `bool isDestroyed() const`. Retorna `true` si l'atribut `shield` és 0 o menor, i `false` en cas contrari
- `ostream& operator<<(ostream &os,const Fighter &f)`. Operador d'eixida que mostra les dades del caça. Per exemple, si un `F-35B` col·locat a la posició (2,3) té com a identificador 27, la seua velocitat és 100, l'atac és 80 i li queda un valor de 36 a l'escut, el que s'ha de mostrar seria:

  Terminal
  ```
  (F-35B 27 [2,3] {100,80,36})
  ```

  Si no tingués posició assignada, simplement mostraria `[-,-]` en lloc de `[2,3]` (l'operador d'eixida de `Coordinate` s'encarregarà d'això). No s'ha d'afegir cap salt de línia al final.

3.3. AircraftCarrier

Aquesta classe permet gestionar portaavions, que tindran diversos atributs (privats):

- `name`: nom del portaavions
- `wins`: victòries obtingudes pels caces del portaavions

[1] Amb aquesta fórmula, el caça més ràpid tindrà un valor més alt de u i per tant tindrà més probabilitat d'atacar.

- `losses`: derrotes dels caces del portaavions
- `fleet`: flota de caces del portaavions (has d'utilitzar un `vector<Fighter *>` per emmagatzemar els caces)

A part dels getters trivials, els mètodes d'aquesta classe són:

- `AircraftCarrier(string name)`. Constructor que inicialitza les dades del portaavions (òbviament els atributs *wins* i *losses* s'han d'inicialitzar a 0). Si el nom és una cadena buida s'ha de llençar l'excepció `invalid_argument` amb la cadena `"Wrong name"`
- `vector<Fighter *> getFleet() const`. Retorna la flota de caces del portaavions, s'utilitzarà només en les proves unitàries
- `void addFighters(string fd)`. A partir d'una cadena com `"5/F-35B:12/F-18:3/A6:2/F-35B"`, ha de construir els caces indicats i afegir-los a la flota de caces del portaavions. El nombre de caces apareixerà al principi, separat del tipus per "/", i si hi ha més d'un tipus de caça es separaran amb ":". A la cadena de l'exemple, es crearien 5 `F-35B`, 12 `F-18`, 3 `A6` i 2 `F-35B`. Es pot assumir que la cadena no tindrà errors, i per tant no cal comprovar-ho. Si la cadena és buida, el mètode no farà res
- `void updateResults(int r)`. Ha d'actualitzar els valors de *wins* o *losses* en funció del valor de l'argument *r*; si val 1, s'incrementarà *wins*, si val -1 s'incrementarà *losses*. Si *r* té qualsevol altre valor, el mètode no farà res
- `Fighter *getFirstAvailableFighter(string type) const`. Retorna el primer caça (no destruït i no posicionat al tauler) de la flota del tipus indicat per l'argument *type*; si *type* és una cadena buida, retorna el primer caça (no destruït i no posicionat) de qualsevol tipus. Si no hi ha caces no destruïts o directament no hi ha caces, retorna `NULL` (o `nullptr`)
- `int purgeFleet()`. Esborra de la flota els caces destruïts, i retorna el nombre de caces esborrats
- `void showFleet() const`. Mostra la flota completa per l'eixida estàndard, mostrant en cada línia un caça de la flota; si el caça ha estat destruït, al final de la línia s'afegirà la cadena `"(X)"` per indicar que s'ha destruït, com en aquest exemple:

  Terminal

  ```
  (F-35B 24 [-,-] {100,80,-24}) (X)
  (F-35B 25 [1,3] {100,80,80})
  (F-35B 26 [-,-] {100,80,80})
  (F-35B 27 [-,-] {100,80,80})
  (F-35B 28 [-,-] {100,80,80})
  ```

  S'han de mostrar tots els caces de la flota, incloent-hi els destruïts i els posicionats; si la flota no té caces, no farà res
- `string myFleet() const`. Retorna una cadena amb el format que admet el mètode `addFighters` amb els caces (destruïts o no) de la flota. Per exemple, si un portaavions té 7 `F-35B` i 1 `Super Hornet`, retornaria la cadena `"7/F-35B:1/Super Hornet"` (o la cadena `"1/Super Hornet:7/F-35B"`, segons si el primer caça és un `F-35B` o un `Super Hornet`). Si la flota no té caces, retorna una cadena buida. Cal tenir en compte que s'han de mostrar els caces en l'ordre que tenen a `fleet` i que s'han d'acumular tots els caces del mateix tipus; si per exemple un portaavions té a la seua flota 3 `F-35B`, 2 `F-18`, 4 `F-35B`, 5 `Super Hornet` i 2 `F-35B`, la cadena a retornar ha de ser `"9/F-35B:2/F-18:5/Super Hornet"` perquè el primer tipus de caça que apareix a la flota és el `F-35B`, després el `F-18` i finalment el `Super Hornet`
- `ostream& operator<<(ostream &os,const AircraftCarrier &a)`. Operador d'eixida que mostra les dades del portaavions. Per exemple, si un portaavions anomenat USS Abraham Lincoln ha aconseguit 35 victòries i ha tingut 10 derrotes, i té una flota de 12 `F-35B` i 7 `F-18`, es mostraria el següent, sense afegir salt de línia al final:

Terminal

```
Aircraft Carrier [USS Abraham Lincoln 35/10] 12/F-35B:7/F-18
```

3.4. Board

Aquesta classe representa el tauler quadrat en què es desenvoluparà el joc, i té dos atributs:

- `size`: mida del tauler (ha de ser sempre positiva). Les coordenades dins del tauler aniran de 0 a *size-1*
- `board`: matriu quadrada de punters a caces per emmagatzemar els caces en posicions del tauler. L'has de declarar com:

```
vector<vector<Fighter *>> board;
```

Els mètodes d'aquesta classe són:

- `Board(int size)`. Constructor que inicialitza les dades del tauler. Si el valor de `size` no és més gran que `0` s'ha de llançar l'excepció `invalid_argument` amb la cadena `"Wrong size"`; si és correcte, s'ha de crear la matriu quadrada de mida `size` i inicialitzar tots els punters a `NULL` o `nullptr` (més endavant s'hi emmagatzemaran caces)
- `Fighter *getFighter(Coordinate c) const`. *Getter* que retorna el contingut del tauler a la posició indicada, que serà el caça que l'ocupa, o `NULL` (`nullptr`) si no hi ha res
- `bool inside(Coordinate c) const`. Retorna `true` si la coordenada està dins del tauler i `false` en cas contrari. Les coordenades estan dins del tauler si les components de la coordenada estan entre 0 i *size-1* (ambdós valors inclosos)
- `int launch(Coordinate c,Fighter *f)`. Intenta col·locar un caça en una posició del tauler (si no està ja posicionat). Òbviament, si la coordenada no està dins del tauler no farà res, igual com tampoc ho farà si la posició està ocupada per un altre caça del mateix portaavions. Tanmateix, si la posició està ocupada per un caça d'un altre portaavions (un caça enemic), el nou caça atacarà i lluitarà amb el que ocupava aquesta posició. Un cop acabada la lluita (en què sempre guanya un dels caces i l'altre resulta destruït), es quedarà a la posició el caça guanyador, i el caça perdedor reinicialitzarà la seua posició. Finalment, si la posició estava buida, el caça es col·locarà en aquesta posició (actualitzant també la posició del caça amb el *setter* corresponent)

  Per exemple, si tenim a la posició `[2,3]` del tauler un `SF-1` i volem llançar un caça atacant `F-35B` a aquesta posició, el `F-35B` lluitarà amb el `SF-1`, i poden passar dues coses:

  - si guanya el `F-35B` se li assignarà aquesta posició del tauler (i s'actualitzarà la seua posició), i al `SF-1` se li reinicialitzarà la posició
  - si guanya el `SF-1` cap dels dos caces canviarà de posició (el `F-35B` no ha de tenir posició per ser llançat, es queda igual)

  El mètode ha de retornar el resultat de la lluita amb el caça enemic: un 1 si guanya el caça atacant o un -1 si guanya l'enemic (el que estava al tauler). En qualsevol altre cas (ja estava al tauler, o no hi ha caça, o és aliat, o la coordenada no està dins del tauler) el mètode retornarà 0.

  Finalment, si el mètode retorna el resultat de la lluita, aquest valor s'ha d'utilitzar en una altra part del codi (p.ex. al prac3.cc) per actualitzar les estadístiques de victòries/derrotes dels portaavions de tots dos caces.

4. Programa principal

El programa principal es troba al fitxer `prac3.cc` publicat al Moodle de l'assignatura. És simplement un exemple de com es podrien utilitzar les classes que has creat en un programa. Pots modificar-lo com vulgues per provar les teues classes, no s'ha de lliurar amb la resta de fitxers.

Aquest fitxer conté codi per crear diversos portaavions i caces, i per simular una batalla al tauler:

```
#include <iostream>
#include "AircraftCarrier.h"
#include "Board.h"

using namespace std;

int main(){
    srand(888);

    Board board(5);  // 5x5

    AircraftCarrier one("USS Acme");
    one.addFighters("3/F-35B:2/F-18:3/A6");

    AircraftCarrier two("Spectra One");
    two.addFighters("4/SF-1:3/SB-3:2/SI-6B");

    // position some fighters on the board
    Coordinate c1(2,3);
    Fighter *f1 = one.getFirstAvailableFighter("F-18");
    board.launch(c1,f1);

    Coordinate c2(3,2);
    Fighter *f2 = two.getFirstAvailableFighter("");
    board.launch(c2,f2);

    cout << "After launching one fighter of each aircraft:" << endl;
    cout << one << endl;
    one.showFleet();
    cout << two << endl;
    two.showFleet();

    Fighter *f3 = one.getFirstAvailableFighter("");
    int result = board.launch(c2,f3);
    if (result != 0){
        one.updateResults(result);
        two.updateResults(-result);
    }
    cout << "After a fight between two fighters:" << endl;     cout << one << endl;
    one.showFleet();
    cout << two << endl;
    two.showFleet();

    return 0;
}
```

L'eixida seria:

```
After launching one fighter of each aircraft:
Aircraft Carrier [USS Acme 0/0] 3/F-35B:2/F-18:3/A6
(F-35B 1 [-,-] {100,80,80})
(F-35B 2 [-,-] {100,80,80})
(F-35B 3 [-,-] {100,80,80})
(F-18 4 [2,3] {100,80,80})
(F-18 5 [-,-] {100,80,80})
```

```
(A6 6 [-,-] {100,80,80})
(A6 7 [-,-] {100,80,80})
(A6 8 [-,-] {100,80,80})

Aircraft Carrier [Spectra One 0/0] 4/SF-1:3/SB-3:2/SI-6B
(SF-1 9 [3,2] {100,80,80})
(SF-1 10 [-,-] {100,80,80})
(SF-1 11 [-,-] {100,80,80})
(SF-1 12 [-,-] {100,80,80})
(SB-3 13 [-,-] {100,80,80})
(SB-3 14 [-,-] {100,80,80})
(SB-3 15 [-,-] {100,80,80})
(SI-6B 16 [-,-] {100,80,80})
(SI-6B 17 [-,-] {100,80,80})

After a fight between two fighters:
Aircraft Carrier [USS Acme 1/0] 3/F-35B:2/F-18:3/A6
(F-35B 1 [3,2] {100,80,4})
(F-35B 2 [-,-] {100,80,80})
(F-35B 3 [-,-] {100,80,80})
(F-18 4 [2,3] {100,80,80})
(F-18 5 [-,-] {100,80,80})
(A6 6 [-,-] {100,80,80})
(A6 7 [-,-] {100,80,80})
(A6 8 [-,-] {100,80,80})

Aircraft Carrier [Spectra One 0/1] 4/SF-1:3/SB-3:2/SI-6B
(SF-1 9 [-,-] {100,80,-3}) (X)
(SF-1 10 [-,-] {100,80,80})
(SF-1 11 [-,-] {100,80,80})
(SF-1 12 [-,-] {100,80,80})
(SB-3 13 [-,-] {100,80,80})
(SB-3 14 [-,-] {100,80,80})
(SB-3 15 [-,-] {100,80,80})
(SI-6B 16 [-,-] {100,80,80})
(SI-6B 17 [-,-] {100,80,80})
```