Memòria Projecte AirTrafficController

Index

Presentació

AirTrafficController és un petit projecte que intenta simular, més o menys, el comportament dels avions. Essencialment es tracta d'un "gestor d'avions" que permet tant afegir avions a l'espai aeri, com modificar-los. L'objectiu del projecte és aprendre a utilitzar les classes i objectes de la programació orientada a objectes, per a fer-ho més interactiu per a l'usuari s'ha enfocat el projecte cap a un petit simulador.
He escollit c# com a llenguatge de programació per la seva similitud amb Java i el comodíssim IDE de visual studio. També he utilitzat les llibreries de Microsoft XNA per a dibuixar en pantalla i el framework MonoGame, que inclou XNA

Classes i mètodes més importants i el seu funcionament

Game1

Comencem amb la classe **Game1**, aquesta classe és la classe principal (a part de la de Program) que gestiona el joc. Conté tots els formularis i classes secundaries del joc, com per exemple el formulari principal o el mapa. Aquesta classe és heredada de la classe **Game**, una classe de la llibreria del framework de **XNA**. La classe Game té varis mètodes principals com pot ser:

- LoadContent
    - Aquí és on es **carreguen** els icones i d'altres objectes "pesats" per a la memòria i CPU
- Update
    - Mètode que es crida constantment, depèn del processador i és on es troba la **lògica** del joc/simulador
    - Per a facilitar l'actualització de les dades, he creat un mètode que es diu "**updateByOneSecond**" que es crida cada 1 segon i és on s'actualitzen tots els avions.
- Draw
    - Mètode que s'encarrega de cridar-se de forma constant i sense interrupcions, ja que és qui "**dibuixa**" en pantalla la informació.

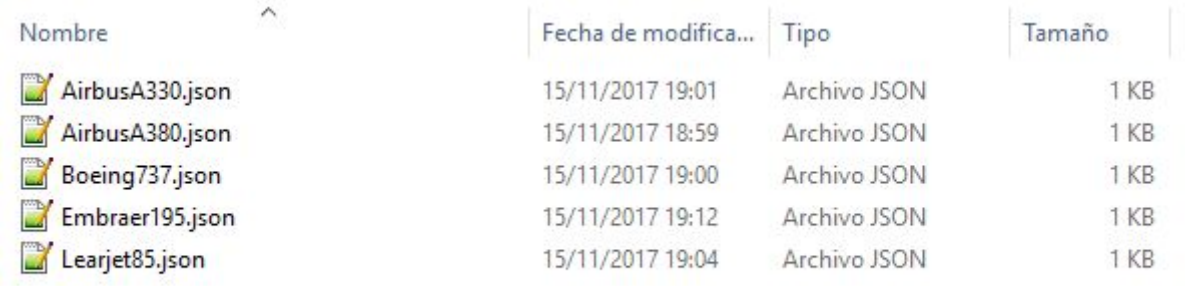
La classe Game1 té mètodes accessors bàsics, no estan tots definits ja que no són necessaris (no tindrien cap utilitat).

Mètode LoadContent

Aquí s'instancien els objectes del joc, com per exemple:

- SpriteBatch, encarregat de dibuixar en pantalla, s'utilitza al mètode Draw
- Stats, classe per a guardar informació i fer estadístiques
- Map, el mapa del simulador
- Les icones dels avions

- Càrrega dels presets d'avions des d'arxius JSON externs



Mètode Update

Al mètode **Update** hi ha definides vàries funcionalitats:

- Si es fa clic dret només, es crea un avió amb dades aleatòries i direcció aleatòria.
- Si es fa clic dret prement alguna de les tecles W A S D, es crea un avió amb la direcció de la tecla premuda.
- Es fa un càlcul dels FPS actuals amb la classe **FrameCounter**, feta per un usuari de StackOverflow i creditada a la mateixa classe.
- Si es fa clic esquerra a sobre d'un avió es cridarà al mètode click de l'avió.
- S'actualitza el gestor de notificacions.
- Es guarda l'estat del ratolí i del teclat per a comprovar els inputs.
- Es crida al mètode updateByOneSecond que a la vegada crida al mètode Update del mapa

Mètode Draw

Aquest mètode s'encarrega de dibuixar a la pantalla i de representar visualment cada objecte. Es crida constantment i només dibuixa. Es calcula l'avió més proper al ratolí per tal de donar feedback a l'usuari per quan passi el ratolí per sobre de qualsevol avió (hovering). Es crida al mètode Draw del mapa, es mostren els FPS i es dibuixen les notificacions.

Map

La classe map s'encarrega de la gestió dels avions. Els mètodes principals s'assemblen als de la classe principal **Game1** però no fan certament el mateix.

Mètode Update

L'actualització del mapa consta de vàries fases:

1. Actualitzar cadascun dels avions que hi es trobi dins de la variable **airplanes**(Llista de iAirplanes)
2. Comprovar que tots els avions segueixen dins del límit del mapa, els que es troben fora, es guarden en una llista temporal i després s'esborren.
3. Comprova els xocs: (mètode **checkCrash**)
    a. Primer es comprova el xoc per altura, es té en compte:
        i. El tren d'aterratge baixat o no
        ii. La velocitat menor que la variable **landingMaxSpeed** (per defecte 50)
        iii. Depenent del cas, el xoc pot ser mortal o parcialment mortal
    b. A continuació es comprova el xoc amb d'altres avions
        i. Es calcula el radi amb els altres avions (tots amb tots).
        ii. El radi té en compte les 3 dimensions, X, Y, Z (en aquest cas: x,y,altura)
    c. El mètode retorna un true si es xoca o un false si tot és correcte
    d. Si l'avio es xoca, es guarda en una llista i s'esborra l'avió a continuació.
4. Es comproven els perills:
    a. Perill de col·lisió
        i. Es calcula el radi en 3 dimensions de tots els avions i si es menor que la variable **distanceCollisionDangerRadius** es notifica als dos avions que estan en perill.
    b. Perill d'altura
        i. Es comprova que l'altura de l'avió sigui superior a la variable **altitudeDanger** i en cas de ser inferior se li notifica a l'avio amb un flag.

Mètode Draw

Es calcula l'area que ocupa la finestra i es dibuixa un rectangle amb certs marges. També es criden als mètodes Draw de tots els avions de la llista **airplanes**.

Airplane

La classe airplane representa a l'avió estàndard, disposa de mètodes accessors setters i getters, definits a la interfície iAirplane, a més d'un mètode inicialitzador anomenat **Initialize**, que té com a arguments totes les variables possibles.

Mètode Initialize

Permet l'inicialització de les variables de l'avió. També permet l'edició d'aquest, reemplaçant les dades anteriors.

Mètode Draw

Aquest mètode és el més complex en quant a càlculs, s'encarrega de dibuixar l'avió en pantalla i més informació com per exemple els perills, els passos que segueix són els següents:

1. Es fa un càlcul per saber on es dibuixarà l'avió:
    a. Gràcies a la classe **utilDraw** es fa un mapping (escalat) de la posició de l'avio dins del mapa al tamany de la finestra.
2. Si té el flag activat de hovering, es dibuixa un cercle al voltant
    a. Utilitzo una llibreria anomenada **Primitives2D** creada per un usuari d'internet, els credits es troben a la mateixa classe.
3. Es dibuixa l'icona de l'avió
    a. Es calcula l'angle de rotació segons la direcció que té l'avió
    b. Es calcula l'origen del sprite segons el seu tamany
    c. L'escala de l'icona depèn del tamany del mapa
4. Si el flag **drawInfo** està activat es dibuixa la informació de l'avió al costat
5. A continuació es dibuixen els diferents DANGERS:
    a. Perill de col·lisió amb altres avions
    b. Perill de caiguda
    c. Perill d'altitud
    d. Perill de tren d'aterratge baixat
6. Es reinicia el flag hover

Mètodes getters i setters

Útils per a demanar informació i manipular l'avió des de fora.

notificationsManager

Petit gestor de notificacions útil per a debugar i útil per a l'usuari.

Mètode addNotification i sobrecàrregues

El mètode consisteix en afegir un objecte del tipus **notification** a la llista **notifications** del gestor de notificacions. Hi ha varis mètodes que permeten afegir una notificació ometent varis arguments deixant amb el mínim, que és el missatge.

Mètode Update

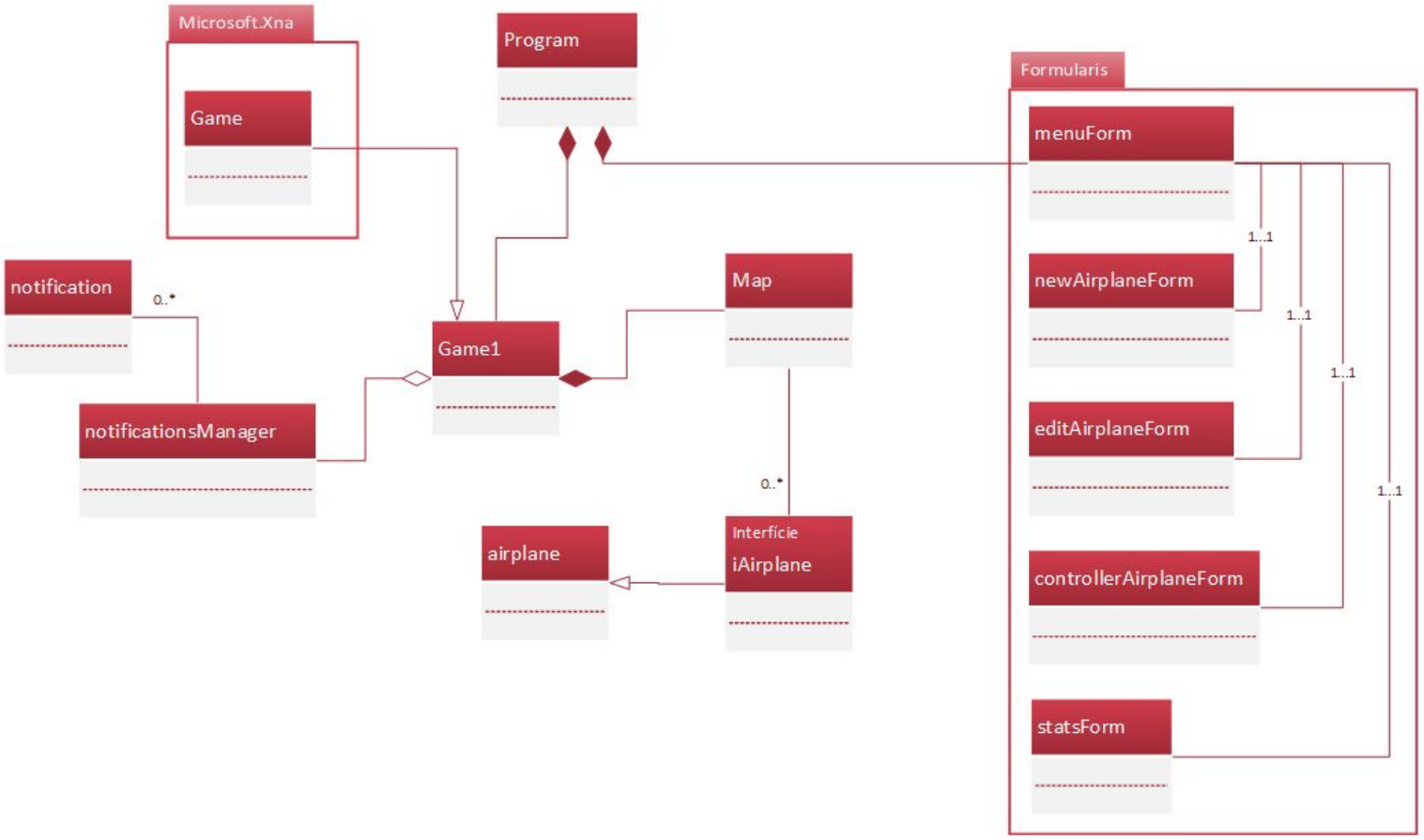
S'encarrega de mantenir l'ordre les notificacions, si el seu temps màxim ja ha passat, s'esborren.

Mètode Draw

S'encarrega de dibuixar les notificacions una per una a la posició que pertoca segons la quantitat de notificacions acumulades.

UML

A continuació una aproximació en UML de l'esquema general del simulador. Només s'hi representen les classes més importants i determinants del programa, llibreries i altres classes auxiliars no estan incloses. Les propietats i mètodes també s'han obviat degut a la gran quantitat que hi podem trobar en cadascuna de les classes.

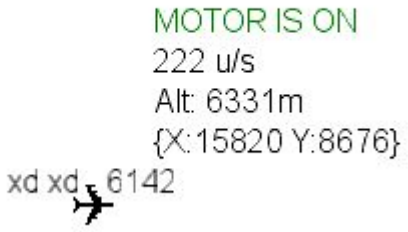


Joc de proves

Per tal de provar les característiques i la lògica del simulador he programat varis TEST que permeten comprovar el funcionament de les mecàniques. Es troba al menú de “**TESTS**” del menú principal.

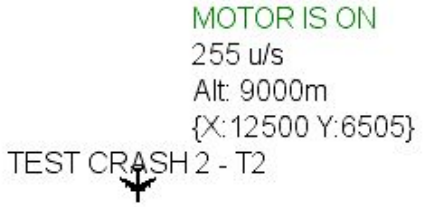
Random airplane

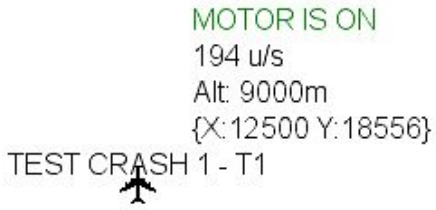
Aquesta opció crearà un avió amb propietats aleatòries.



Test collision

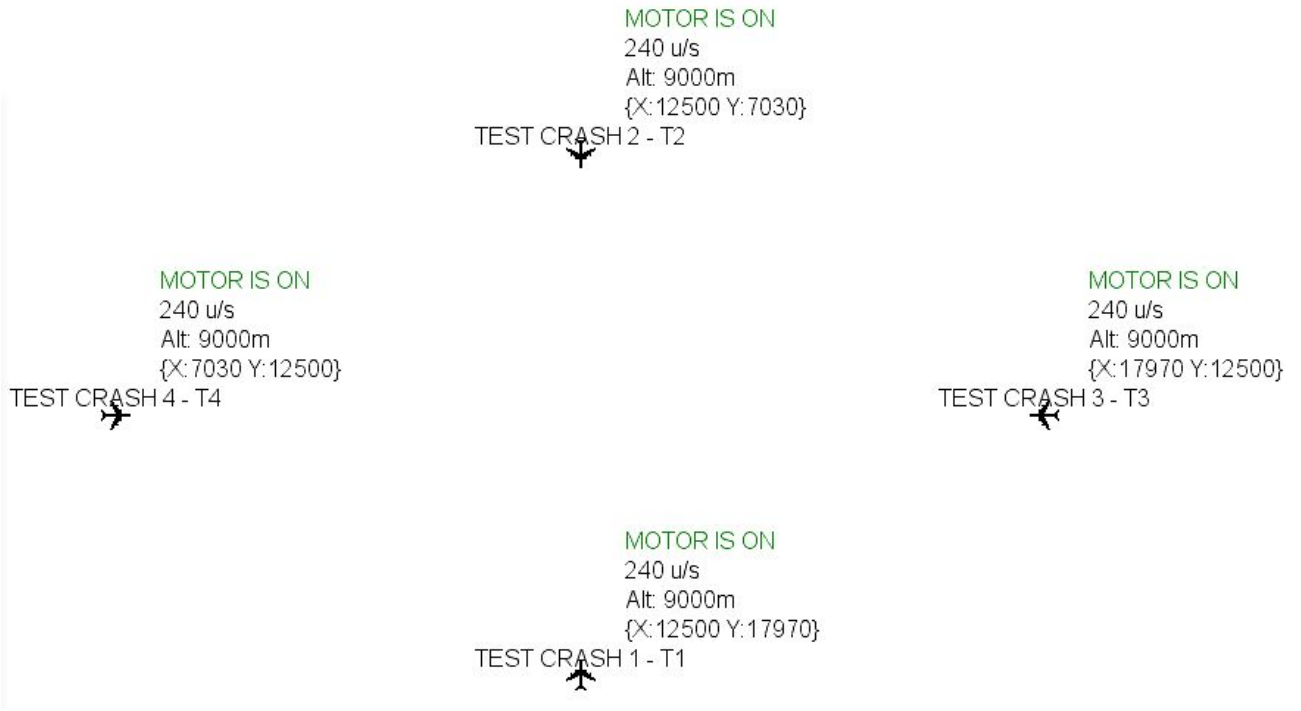
Aquesta opció crearà dos avions amb direccions oposades (per simular xoc) a la mateixa altitud i a la mateixa coordenada X. L’objectiu de la prova és veure com col·lisionen i què passa.





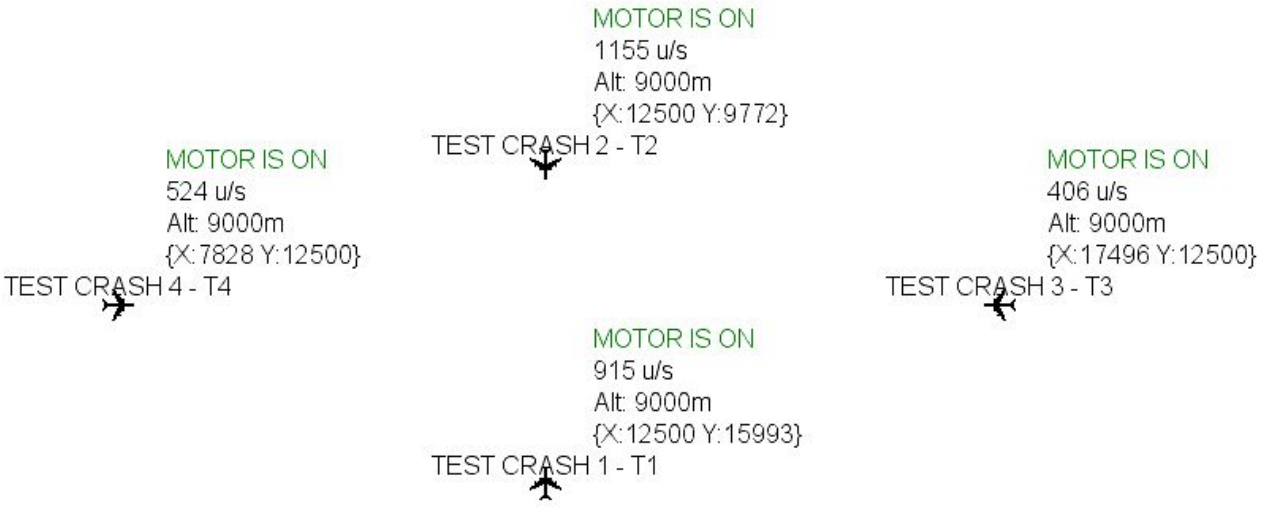
Test multiple collision, same speed/acceleration

El mateix que l'anterior però amb 4 avions a punt de xocar, amb la mateixa velocitat i acceleració.



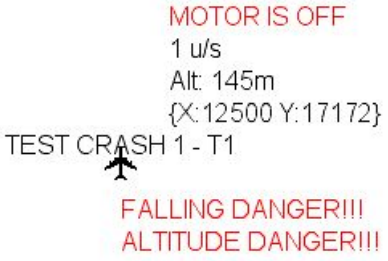
Test multiple collision, random speed/acceleration

El mateix que l'anterior però amb velocitat i acceleració aleatòria. Amb aquest test podem veure com es van xocant poc a poc els avions.



Test collision with ground

Un avió amb el motor apagat, amb perill de caiguda (acceleració vertical negativa) amb una velocitat aleatòria i amb el tren d'aterratge pujat. Segons la velocitat en que toqui el terra, hi haurà un accident catastròfic (100% víctimes mortals) o accident mig solventat.



Airplane T1 has crashed with the ground. 138 people have died. Congratulations.

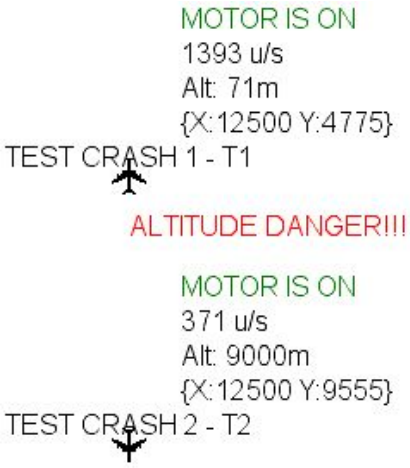
Test NO collision with airplanes (altitude)

El mateix test que el de col·lisions amb dos avions però amb altituds diferents, en teoria no s'haurien de xocar mai.

Els dos avions es creuen i no hi ha cap alerta de col·lisió entre avions.

ALTITUDE DANGER!!!

Els avions es creuen i segueixen el seu rumb.



Conclusió

És interessant replicar la realitat en programació, fa veure lo complicat que és tot en realitat. Gràcies a aquest petit simulador he pogut apreciar de primera mà lo difícil que és fer un simulador, malgrat que AirTrafficController no és fidel del tot a la realitat, he intentat fer-lo lo més real possible amb un entorn gràfic i senzill però complert. AirTrafficController és dels primers “jocs” que faig, també és el primer joc que hi incorporo una interfície gràfica on poder veure el joc en viu “movent-se” per la pantalla i no per la consola.