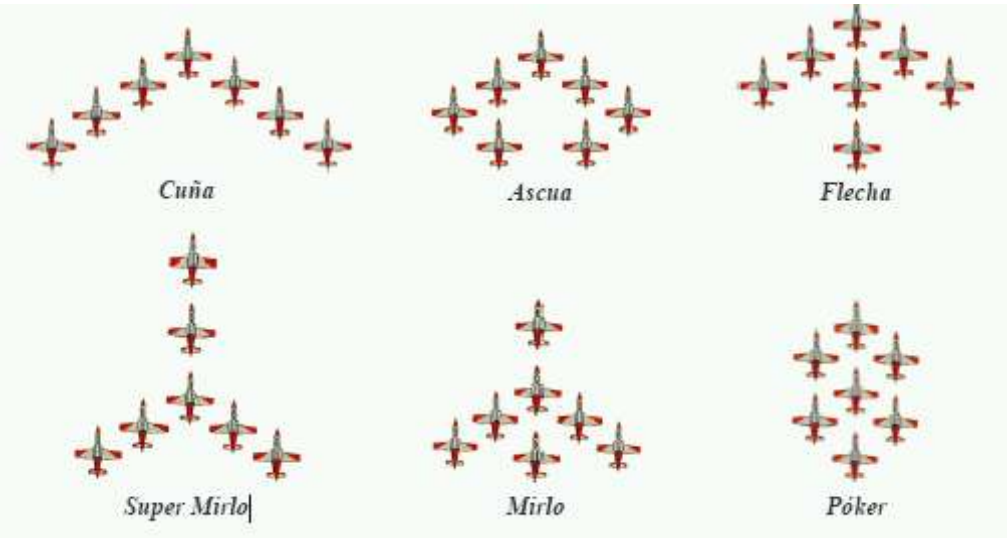
Índex de continguts

Enunciat

Dissenyar un planificador de moviments que determini les trajectòries dels set avions que integren la “Patrulla Àguila” de l'exèrcit de l'aire, de manera que puguin realitzar canvis de formació entre qualsevol de les sis formacions següents:



Donat que tots els moviments s’han de realitzar sobre el pla definit per la formació, es considerarà que cada avió només té dos graus de llibertat de translació. El programa realitzat ha de planificar el canvi entre qualsevol parell de formacions i simular gràficament la seva execució. Els canvis de formació es realitzaran en el mínim temps possible, per la qual cosa s’han de minimitzar els moviments dels avions.

Lliurament

La pràctica es realitzarà en grups de dues persones com a màxim i serà lliurada abans de l’examen de l’assignatura. El lliurament consistirà en una demostració pràctica al laboratori i una documentació de la implementació realitzada. Aquesta documentació es remetrà dins d’un sobre amb els noms dels estudiants clarament visibles, i amb totes les pàgines numerades i sense grapar.

A més de la documentació impresa, s’enviarà a l’adreça `magarcia@etse.urv.es` un fitxer `.tar` comprimit amb la comanda `gzip` de Unix, el qual contindrà un bolcat de tot el directori que conté la pràctica.

Disseny

Amb l'objectiu de planificar el moviments dels diferents avions, s'utilitzara un Espai de Configuracions Temporal (ECT) de 2 graus de llibertat (x, y) més la dimensió del temps.

L'espai de configuracions es dividirà en cel·les quadrades de tamany fix i es considerarà que un avió hi cap en una cel·la, d'aquesta forma s'evita haver de fer engreixat dels obstacles. Els obstacles consisteixen en trajectòries d'avions anteriorment planificats.

Les formacions possibles dels avions es discretitzen per tal d'indicar quina cel·la ocupa cada avió, de forma que l'espai total estarà format per 7 x 7 cel·les i cada avió te assignat un identificador únic (0..6). Els identificadors indiquen l'ordre en el que es planificaran les trajectòries (el primer serà el 0, l'últim el 6).

Cunya	***Ascua***	***Fletxa***
- - - 0 - - - - - 1 - 2 - - - 3 - - - 4 - 5 - - - - - 6 -	- - 0 - - - - - 1 - 2 - - - 3 - - - 4 - - - 5 - 6 -	- - 0 - - - - - 1 - 2 - - - 3 - 4 - 5 - - - - - - - - - - - 6 - - - - - - - - - - - - - - - - - -
SuperMirlo	**Mirlo**	**Poker**
- - 0 - - - - - - - - - - - - - 1 - - - - - - - - - - - - - 2 - - - - - 3 - 4 - - - 5 - - - 6 - -	- - 0 - - - - - - - - - - - - - 1 - - - - - 2 - 3 - - - 4 - 5 - 6 - - - - - - - - - - - - - - - -	- 0 - - - - - 1 - 2 - - - - - 3 - - - - - 4 - 5 - - - - - 6 - - - - - - - - - - - - - - - - - - -

Per tal de trobar les trajectòries mínimes s'ha implementat l'algorisme Dijkstra amb MinHeap com optimització.

Estructura per l'Espai de Configuracions Temporal

L'espai de configuracions temporal es troba representat per una taula de 3 dimensions:

- Dimensió 1 (x): 7 posicions
- Dimensió 2 (y): 7 posicions

- Dimensió 3 (temps): 20 posicions

Cada posició contindrà una estructura amb la següent informació:

- **Avió**: Id de l'avió que ocupa aquesta posició (si es troba a -1, no hi ha cap).
- **NodeAnterior**: Apunta al node del que provenim. S'utilitza per reconstruir el camí trobat i assignar els valors al ECT.
- Tractat: Irrellevant per la solució final. Indica que ha sigut tractat a la generació del graf i inicialització del Heap, conté id de l'avió actual si ha sigut tractat.
- Pendent: Irrellevant per la solució final. Indica que la posició esta pendent durant la cerca del camí mínim entre la posició inicial i final, conté id de l'avió actual si es troba pendent.

Successors

Per tal de trobar una trajectoria des d'una posició inicial, s'han d'anar generar posicions successores. Per realitzar aquesta generació es té en compte:

- Un successor te un instant de temps superior al seu pare.
- El temps màxim es troba acotat a 20.
- Una posició només podrà ser successora si no hi ha cap altre avió situat.
- Quan la posició actual es trobi al límit de l'espai, s'evita la generació de successors fora dels límits.
- Si la posició successora en el temps actual es troba ocupada per l'avió A i la posició actual en el temps següent es troba ocupada pel mateix avió A, no es considerarà com aquesta posició successora per evitar creuaments.

Algorisme general

La formació inicial serà sempre Cunya. Un cop sapiguem quina es la propera formació, es duran a terme els següents passos:

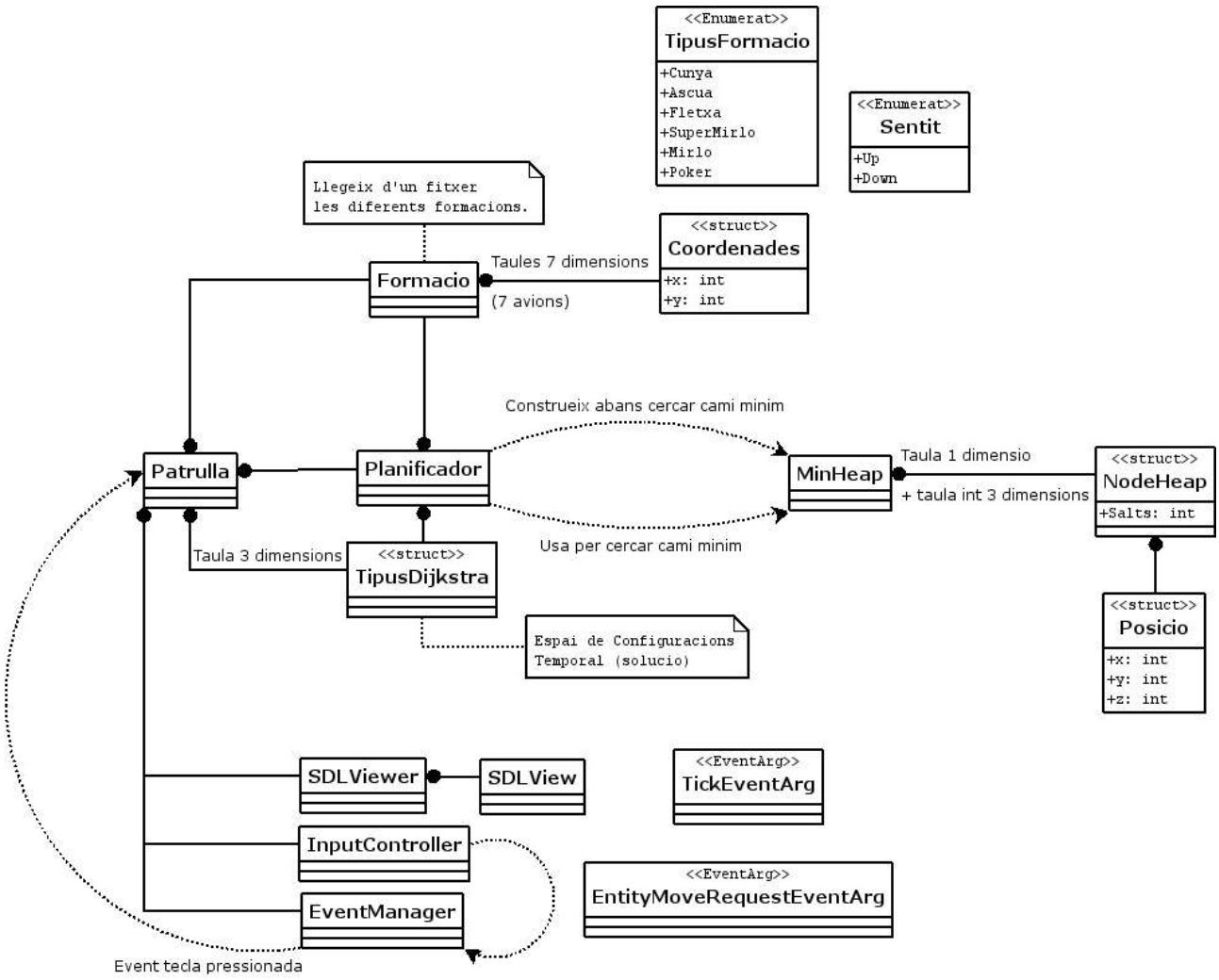
- S'aplica dijkstra per a cada avió
  - Es construeix Graf i Heap
    - Partint de la posició inicial, es generen els successors i es guarden a una Pila
    - Es generen els successors dels successors fins buidar la Pila
  - Busca camí mínim
    - Es tracten els elements del Heap i es van assignant els nombres de salt
    - Si troba posició final, retorna el nombre de salts en el temps i acaba
  - Assigna camí al ECT
    - Assigna camí des de posició inicial fins final
    - Assigna posició estàtica des de posició final fins final del temps màxim (20)

- Si en un moment del temps després d'arribar a la posició final, aquesta posició es troba ocupada per un altre avió ja planificat, realitzem maniobra d'evasió movent temporalment l'avió a una posició buida i retornant a la seva posició final al següent instant de temps. **NOTA:** Es pot veure aquesta situació si pasem de Cunya (1) a Poker (6) i observem l'avió rosa.

Implementació

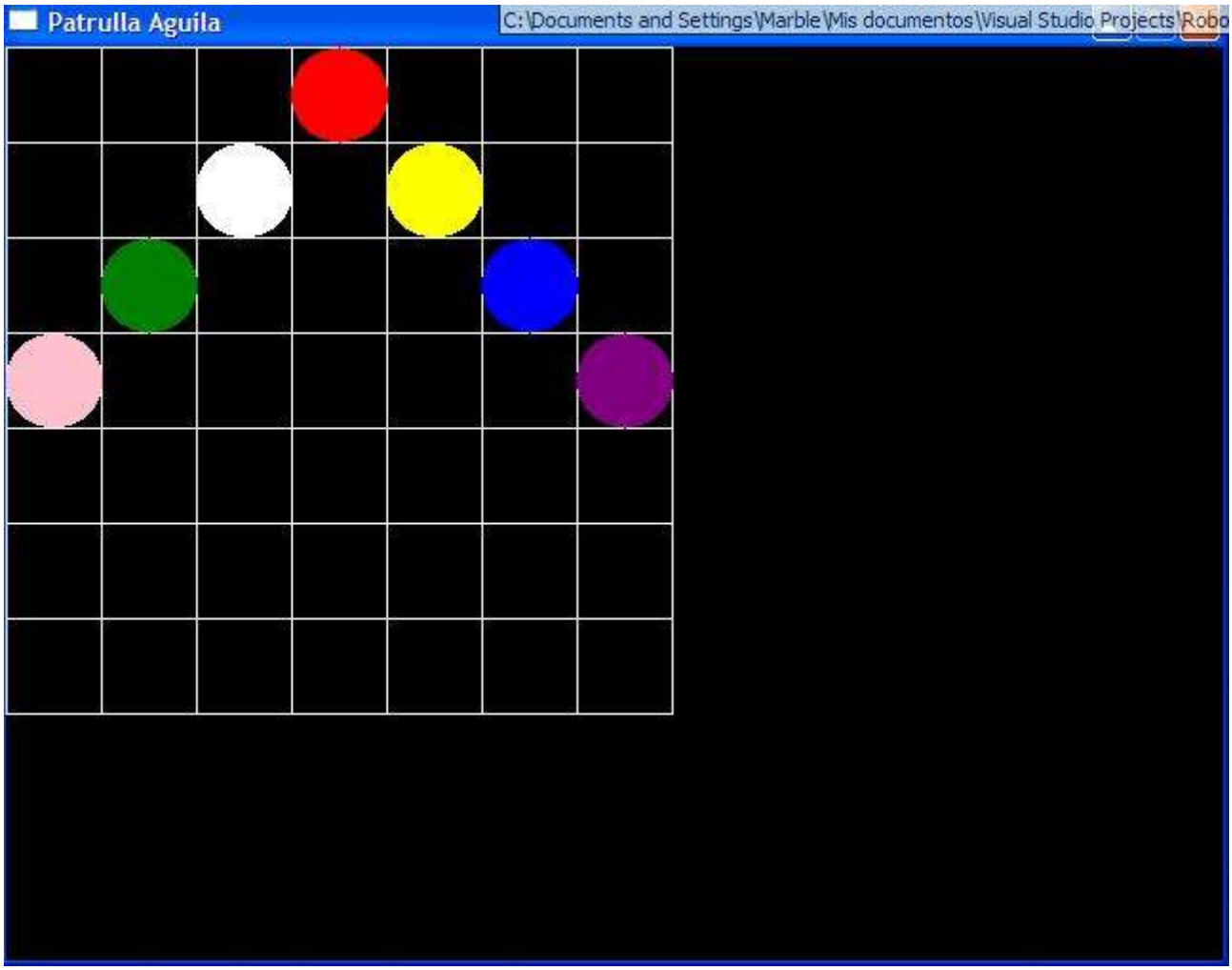
L'aplicació ha sigut desenvolupada amb Visual Studio .NET i les llibreries gràfiques SDL.NET.

El projecte consta de les següents classes i relacions:



La interfície gràfica programada mostra els avions en forma de cercle sobre un taulell quadriculat, cada avió té un color diferent segons el seu identificador:

Identificador	*Color*
0	Red
1	White
2	Yellow
3	Green
4	Blue
5	Pink
6	Purple
7	Plum



Quan l'usuari pressiona les tecles del keypad (teclat númeric) es realitza els càlculs de trajectòries i es mouen els avions amb salts discrets i amb pauses forçades per poder observar el moviment correctament.

Número	*Formació*
1	Cunya
2	Ascua
3	Fletxa
4	SuperMirlo
5	Mirlo
6	Poker

Les formacions es carreguen des d'un fitxer anomenat “formacions.txt”, el qual conté 6 grups (6 formacions) de 7 posicions a on cadascuna correspon a un avió. El format es el següent:

```
# Cunya
0 3
1 2
1 4
2 1
2 5
3 0
3 6
# Ascua
0 2
1 1
1 3
2 0
2 4
3 1
3 3
# Fletxa
0 2
1 1
1 3
2 0
2 2
```

```
2 4
4 2
# Super Mirlo
0 2
2 2
4 2
5 1
5 3
6 0
6 4
# Mirlo
0 2
2 2
3 1
3 3
4 0
4 2
4 4
# Poker
0 1
1 0
1 2
2 1
3 0
3 2
4 1
```

Codi

Patrulla.cs

```
using System;
using System.Threading;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de Class1.
```

```
/// </summary>
class Patrulla
{
        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
                new Patrulla();
        }

        private TipusFormacio formacioAnterior;
        private TipusFormacio formacioActual;
        private Formacio formacions;
        private Planificador planificador;
        private int maxX;
        private int maxY;
        private int maxTemps;
        private TipusDijkstra[,,] ect;
        private static Mutex mut = new Mutex();
        private EventManager eventManager;
        private InputController inputController;
        private SDLViewer sdlViewer;

        public Patrulla()
        {
                maxX = 7;
                maxY = 7;
                maxTemps = 20;
                formacioAnterior = TipusFormacio.Inici;
                formacioActual = TipusFormacio.Inici;

                // ECT
                ect = new TipusDijkstra[maxX, maxY, maxTemps];

                // Carreguem fitxer amb formacions
                formacions = new Formacio();
                formacions.inicialitzarFormacions("../../formacions.txt");

                // Planificador
```

```
                    planificador = new Planificador(maxX, maxY, maxTemps, formacions, ect);

                    // SDL
                    eventManager = new EventManager();
                    eventManager.OnEntityMoveRequestEvent += new
EventManager.EntityMoveRequestEventHandler(this.Moure);
                    inputController = new InputController(eventManager);
                    sdlViewer = new SDLViewer(eventManager);

                    // Formacio inicial
                    formacioActual = TipusFormacio.Cunya;
                    sdlViewer.ShowFormacio(formacions.obtenir_posicions_formacio
(formacioActual));
                    formacioAnterior = formacioActual;

                    // Espera infinita
                    inputController.Run();
            }

            public void Moure(object sender, EntityMoveRequestEventArgs e)
            {
                    if (!mut.WaitOne(0, false))
                    {
                            return;
                    }

                    formacioActual = e.Formacio;

                    // Planificacio i posterior visualitzacio dels moviments
                    // necessaris de la patrulla per passar de formacio_anterior
                    // fins a formacio_actual
                    int maxTemps = planificador.planificarMoviments(formacioAnterior,
formacioActual);
                    TipusCoordenada coord;
                    TipusCoordenada[] coordenades = new TipusCoordenada[7];
                    for(int temps = 0; temps <= maxTemps; temps++)
                    {
                            for(int i=0; i<7; i++)
                            {
                                    posicioAvio(this.ect, temps, i, out coord);
                                    coordenades[i] = coord;
```

```
                                        }
                                        Thread.Sleep(750);
                                        sdlViewer.ShowFormacio(coordenades);
                                }

                                formacioAnterior = formacioActual;

                                mut.ReleaseMutex();
                        }

                        public bool posicioAvio(TipusDijkstra[,,] ect, int temps, int avio, out TipusCoordenada
coord)
                        {
                                bool trobat = false;
                                int i = 0;
                                int j = 0;
                                while(!trobat && i < 7)                 // maxX = 7
                                {
                                        if (ect[i, j, temps].avio == avio)
                                        {
                                                trobat = true;
                                        }
                                        else
                                        {
                                                j++;
                                                if (j == 7)                             //
maxY = 7
                                                {
                                                        i++;
                                                        j = 0;
                                                }
                                        }
                                }

                                coord.x = i;
                                coord.y = j;
                                return trobat;
                        }
                }
}
```

MinHeap.cs

```
using System;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de MinHeap.
        /// </summary>
        public class MinHeap
        {
                private NodeHeap[] taulaHeap;
                private int[,,] hash;
                private int elements;
                private int maxX, maxY, maxTemps, maxElements;

                // Constructor
                public MinHeap(int maxX, int maxY, int maxTemps)
                {
                        this.maxElements = maxX * maxY * maxTemps;
                        taulaHeap = new NodeHeap[maxElements];
                        hash = new int[maxX, maxY, maxTemps];
                        elements = 1;
                        this.maxX = maxX;
                        this.maxY = maxY;
                        this.maxTemps = maxTemps;
                }

                // Retorna el valor mes petit del heap, el cap.
                public NodeHeap obtenirMesPetit()
                {
                        if(elements == 1)
                        {
                                throw new Exception("El MinHeap esta buit!");
                        }
                        else
                        {
                                return taulaHeap[1];
                        }
```

```
	}

	// Inserta un nou node en el Min_Heap.
	public void afegir(NodeHeap node)
	{
		if(elements == maxElements + 1)
		{
			throw new Exception("El MinHeap esta ple!");
		}
		else
		{
			int k = elements;
			elements++;
			bool final = false;
			while(!final && k > 1)
			{
				if(node.salts < taulaHeap[k / 2].salts)
				{
					hash[taulaHeap[k / 2].pos.x, taulaHeap[k / 2].pos.y, taulaHeap[k / 2].pos.z] = k;
					taulaHeap[k] = taulaHeap[k / 2];
					k = k / 2;
				}
				else
				{
					final = true;
				}
			}
			hash[node.pos.x, node.pos.y, node.pos.z] = k;
			taulaHeap[k].pos = node.pos;
			taulaHeap[k].salts = node.salts;
		}
	}

	// Consulta si esta buit o no.
	public bool heapBuit()
	{
		return (elements == 1);
	}

	// Esborra el cap i reordena el heap.
```

```
                public void esborrar()
                {
                        if(heapBuit())
                        {
                                throw new Exception("El Min_Heap esta buit!");
                        }
                        else
                        {
                                int k = 1;
                                bool final = false;
                                while ((k * 2 < elements - 1) && !final)
                                {
                                        int fill = fillMesPetit(k);
                                        if (taulaHeap[fill].salts < taulaHeap[elements -
1].salts)
                                        {
                                                hash[taulaHeap[fill].pos.x,
taulaHeap[fill].pos.y, taulaHeap[fill].pos.z] = k;
                                                taulaHeap[k] = taulaHeap[fill];
                                                k = fill;
                                        }
                                        else
                                        {
                                                final = true;
                                        }
                                }
                                hash[taulaHeap[elements - 1].pos.x, taulaHeap[elements - 1].
pos.y, taulaHeap[elements - 1].pos.z] = k;
                                taulaHeap[k] = taulaHeap[elements - 1];
                                elements = elements - 1;
                        }
                }

                // Cerca la posicio del ECT indicada per veure
                // en la corresponent posicio del Min_Heap quin valor hi ha emmagatzemat, en
        // aquest cas el valor guardat es el numero de salts respecte l'origen.
                public int consultarValor(Posicio pos)
                {
                        if (pos.x < 0 || pos.x > maxX || pos.y < 0 || pos.y > maxY ||
                                pos.z < 0 || pos.z > maxTemps)
                        {
```

```
                throw new Exception("Posicio inexistent en el Heap!");
            }
            else
            {
                int index = hash[pos.x, pos.y, pos.z];
                return taulaHeap[index].salts;
            }
        }

        // Modifica el valor d'un element concret del heap i reordena el heap segons el nou valor establert.
        public void canviarValor(Posicio pos, int nouValor)
        {
            if (pos.x < 0 || pos.x > maxX || pos.y < 0 || pos.y > maxY ||
                pos.z < 0 || pos.z > maxTemps)
            {
                throw new Exception("Posicio inexistent en el Heap!");
            }
            else
            {
                int index = hash[pos.x, pos.y, pos.z];
                int min = 1;
                taulaHeap[index].salts = nouValor;
                if (index == 1)
                { // Si l'element modificat es el cap del heap
                    bool fi = valorMajorAFills(index, nouValor, out min);

                    while(!fi)
                    {   // Mentre no estigui correctament ordenat
                        fi = ordenar(ref index, ref min, Sentit.Down, nouValor);

                    }
                }
                else
                { // Si l'element a canviar no es troba al cap del Min_Heap
                    bool avall = false;
                    if ((index * 2) + 1 < elements)
                    {
                        //int val1 = taulaHeap[index * 2].salts;

                        //int val2 = taulaHeap[(index * 2)
```

```
+ 1].salts;
                                                        //if((val1 <= nouValor) || (val2 <=
nouValor))
                                                        if((taulaHeap[index * 2].salts <=
nouValor) || (taulaHeap[(index * 2) + 1].salts <= nouValor))
                                                        {
                                                                avall = true;
                                                        }
                                                }
                                                bool amunt = false;
                                                int val1 = taulaHeap[index / 2].salts;
                                                if(val1 > nouValor)
                                                {
                                                        amunt = true;
                                                }
                                                if(avall)
                                                { // Si cal intercanviar el node amb els fills
perque
                                                        // es mes gran que algun dels fills
                                                        bool fi = valorMajorAFills(index,
nouValor, out min);
                                                        while(!fi)
                                                        { // Mentre no estigui
correctament ordenat
                                                                fi = ordenar(ref
index, ref min, Sentit.Down, nouValor);
                                                        }
                                                }
                                                if(amunt)
                                                { // Si cal intercanviar el node amb el pare
perque
                                                        // es menor que el pare
                                                        bool fi = true;
                                                        //int val1 = taulaHeap[index / 2].
salts;
                                                        //if(val1 > nouValor)
                                                        if(taulaHeap[index / 2].salts >
nouValor)
                                                        {
                                                                fi = false;
                                                                min = index / 2;
```

```
                                                                        }
                                                                        while(!fi)
                                                                        {   // Mentre no estigui ordenat correctament
                                                                                fi = ordenar(ref index, ref min, Sentit.Up, nouValor);
                                                                        }
                                                                }
                                                        }
                                                }
                                        }

                                        // Obte el fill mes petit del pare indicat.
                                        private int fillMesPetit(int k)
                                        {
                                                int petit;

                                                if((k * 2 + 1) == (elements - 1))
                                                {  // Si nomes te fill esquerre
                                                        petit = k * 2;
                                                }
                                                else
                                                {
                                                        if(taulaHeap[k * 2].salts < taulaHeap[k * 2 + 1].salts)
                                                        {
                                                                petit = k * 2;
                                                        }
                                                        else
                                                        {
                                                                petit = k * 2 + 1;
                                                        }
                                                }
                                                return petit;
                                        }

                                        // Compara el valor especificat amb els fills
                                        // de la posicio indicada, i obte si es major o no als valors dels fills, i en
                                        // cas de ser major retorna el fill menor.
                                        private bool valorMajorAFills(int index, int nouValor, out int min)
                                        {
```

```
                bool fi = true;
                int val1 = taulaHeap[index * 2].salts;
                int val2 = taulaHeap[(index * 2) + 1].salts;
                if(val1 < nouValor || val2 < nouValor)
                {
                        fi = false;
                        if(val1 < val2)
                        {
                                min = index * 2;
                        }
                        else
                        {
                                min = (index * 2) + 1;
                        }
                }
                else
                {
                        min = 0;
                }
                return fi;
}

// Permet ordenar el heap durant l'execucio d'un canvi de valor en un elment del Heap.
private bool ordenar(ref int index, ref int min, Sentit sentit, int nouValor)
{

                bool fi = false;
                NodeHeap node = taulaHeap[index];
                taulaHeap[index] = taulaHeap[min];
                taulaHeap[min] = node;
                Posicio posK = taulaHeap[index].pos;
                Posicio posJ = taulaHeap[min].pos;
                hash[posK.x, posK.y, posK.z] = index;
                hash[posJ.x, posJ.y, posJ.z] = min;
                index = min;
                if(sentit == Sentit.Down)
                {
                        if(index == elements)
                        {
                                fi = true;
                        }
```

```
                                        else
                                        {
                                                fi = valorMajorAFills(index, nouValor, out min);
                                        }
                                }
                                else
                                { // sentit == Up
                                        if(index == 1)
                                        {
                                                fi = true;
                                        }
                                        else
                                        {
                                                fi = true;
                                                int val1 = taulaHeap[index / 2].salts;
                                                if (val1 > nouValor)
                                                {
                                                        fi = false;
                                                        min = index / 2;
                                                }
                                        }
                                }
                                return fi;
                        }
                }
}
```

Formacio.cs

```
using System;
using System.IO;

namespace PatrullaAguila
{
        public struct TipusCoordenada
        {
                public int x, y;
        }
```

```
public enum TipusFormacio
{
        Cunya = 0,
        Ascua,
        Fletxa,
        SuperMirlo,
        Mirlo,
        Poker,
        Inici,
        Fi
}

/// <summary>
/// Descripción breve de Formacio.
/// </summary>
public class Formacio
{
        private TipusCoordenada[] formacioCunya = new TipusCoordenada[7];          // 7
avions
        private TipusCoordenada[] formacioAscua = new TipusCoordenada[7];
        private TipusCoordenada[] formacioFletxa = new TipusCoordenada[7];
        private TipusCoordenada[] formacioSuperMirlo = new TipusCoordenada[7];
        private TipusCoordenada[] formacioMirlo = new TipusCoordenada[7];
        private TipusCoordenada[] formacioPoker = new TipusCoordenada[7];

        public Formacio()
        {
        }

        public bool inicialitzarFormacions(string fitxer)
        {
                bool correcte = false;
                string linia;
                try
                {
                        using (StreamReader sr = new StreamReader(fitxer))
                        {
                                linia = sr.ReadLine();
                                linia = sr.ReadLine();
                                for(int i = 0; i < 7; i++) {
                                        formacioCunya[i].x = linia[0] - 48;
```

```
                formacioCunya[i].y = linia[2] - 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
        for(int i = 0; i < 7; i++)
        {
                formacioAscua[i].x = linia[0] - 48;
                formacioAscua[i].y = linia[2] - 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
        for(int i = 0; i < 7; i++)
        {
                formacioFletxa[i].x = linia[0] - 48;
                formacioFletxa[i].y = linia[2] - 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
        for(int i = 0; i < 7; i++)
        {
                formacioSuperMirlo[i].x = linia[0]
- 48;
                formacioSuperMirlo[i].y = linia[2]
- 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
        for(int i = 0; i < 7; i++)
        {
                formacioMirlo[i].x = linia[0] - 48;
                formacioMirlo[i].y = linia[2] - 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
        for(int i = 0; i < 7; i++)
        {
                formacioPoker[i].x = linia[0] - 48;
                formacioPoker[i].y = linia[2] - 48;
                linia = sr.ReadLine();
        }
        linia = sr.ReadLine();
```

```
                                        }
                                        correcte = true;
                                }
                                catch (Exception e)
                                {
                                        Console.WriteLine(e.Message);
                                        correcte = false;
                                }

                                return correcte;
                        }

                        public TipusCoordenada[] obtenir_posicions_formacio(TipusFormacio formacio)
                        {
                                switch(formacio)
                                {
                                        case TipusFormacio.Cunya:
                                                return formacioCunya;
                                        case TipusFormacio.Ascua:
                                                return formacioAscua;
                                        case TipusFormacio.Fletxa:
                                                return formacioFletxa;
                                        case TipusFormacio.SuperMirlo:
                                                return formacioSuperMirlo;
                                        case TipusFormacio.Mirlo:
                                                return formacioMirlo;
                                        case TipusFormacio.Poker:
                                                return formacioPoker;
                                        default:
                                                return null;
                                }
                        }
                }
}
```

Posicio.cs

```
using System;
```

```
namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de Posicio.
        /// </summary>
        public struct Posicio
        {
                public int x, y, z;
        }

        public struct NodeHeap
        {
                public Posicio pos;
                public int salts;
        }

        public enum Sentit
        {
                Up = 0,
                Down
        }
}
```

Planificador.cs

```
using System;
using System.Collections;

namespace PatrullaAguila
{
public struct TipusDijkstra
{
        public int avio;
        public int tractat;
        public int pendent;
        public Posicio nodeAnterior;
}

/// <summary>
```

```
/// Descripción breve de Planificador.
/// </summary>
public class Planificador
{
        private TipusDijkstra[,,] ect;
        private int numMovimentsMajor;
        private Formacio formacions;
        int maxX, maxY, maxTemps;


        public Planificador(int maxX, int maxY, int maxTemps, Formacio formacions_ini, TipusDijkstra[,,]
referencia_ect)
        {
                //etc = new TipusDijkstra[maxY, maxTemps];
                this.ect = referencia_ect;
                this.formacions = formacions_ini;
                this.maxX = maxX;
                this.maxY = maxY;
                this.maxTemps = maxTemps;
        }

        // Es crida externament per
        // planificar les trajectories entre dues formacions d'avions i que finalment
        // deixa en el ECT tots els passos necessaris per realitzar els moviments
        // transitoris entre formacions.
        public int planificarMoviments(TipusFormacio anterior, TipusFormacio actual)
        {
                TipusCoordenada[] coordsAnteriors = formacions.obtenir_posicions_formacio(anterior);
                TipusCoordenada[] coordsActuals = formacions.obtenir_posicions_formacio(actual);
                inicialitzarETC();
                bool camiInexistent = false;
                int numAvio = 0;
                numMovimentsMajor = 0;
                while(numAvio < 7 && !camiInexistent) // 7 avions
                {
                        camiInexistent = dijkstra(coordsAnteriors[numAvio], coordsActuals
[numAvio], numAvio);
                        numAvio++;
                }
                if(camiInexistent)
                {
```

```
                    return -1;
            }
            else
            {
                    return numMovimentsMajor;
            }
}

// Inicialitza els valors del ECT abans de aplicar el calcul planificador de canvi d'una formacio a una altra.
public void inicialitzarETC()
{
            for(int i = 0; i < maxX; i++)
            {
                    for(int j = 0; j < maxY; j++)
                    {
                            for(int k = 0; k < maxTemps; k++)
                            {
                                    ect[i, j, k].avio = -1;
                                    ect[i, j, k].tractat = -1;
                                    ect[i, j, k].pendent = -1;
                            }
                    }
            }
}

// Obte el nombre minim de moviments per arribar de la posicio inicial a la posicio final.
private     int calcularSaltsCamiMinim(TipusCoordenada origen, TipusCoordenada desti)
{
            int num_salts;
            int diferenciaX = 0;
            int diferenciaY = 0;
            int xini = origen.x;
            int yini = origen.y;
            int xfi = origen.y;
            if(xini > desti.x)
            {
                    while (xini > desti.x)
                    {
                            diferenciaX++;
                            xini--;
                    }
```

```
                }
                else
                {
                        while (xini < desti.x)
                        {
                                diferenciaX++;
                                xini++;
                        }
                }
                if(yini > desti.y)
                {
                        while (yini > desti.y)
                        {
                                diferenciaY++;
                                yini--;
                        }
                }
                else
                {
                        while (yini < desti.y)
                        {
                                diferenciaY++;
                                yini++;
                        }
                }
                num_salts = diferenciaX + diferenciaY;
                return num_salts;
        }

        // Crea un struct Posicio
        private Posicio assignar_posicio(int x, int y, int z)
        {
                Posicio pos;
                pos.x = x;
                pos.y = y;
                pos.z = z;
                return pos;
        }

        // A partir d'una posicio determinada del ECT retorna un vector amb tots els possibles successors tenint en
compte
```

```
// els limits del ECT i els obstacles existents actualment en el ECT.
private void calcularSuccessors(Posicio posActual, ArrayList successors)
{
        int x = posActual.x;
        int y = posActual.y;
        int temps = posActual.z;
        if((x > 0) && (temps < maxTemps - 1))
        { // Si no es la primera posicio de x
                if((ect[x - 1, y, temps + 1].avio == -1))
                {
                        if ((ect[x, y, temps + 1].avio == -1) || (ect[x - 1, y, temps].avio
== -1))
                        {
                                successors.Add(assignar_posicio(x - 1, y, temps
+ 1));
                        }
                        else
                        {
                                // Si no hi ha creuament
                                if(ect[x + 1, y, temps].avio != ect[x, y, temps +
1].avio)
                                {  // Si la posicio horitzontal anterior esta lliure )
{
                                        successors.Add(assignar_posicio(x
-1, y, temps + 1));
                                }
                        }
                }
        }
        if((x < maxX - 1) && (temps < maxTemps - 1))
        { // Si no es la ultima posicio de x
                if((ect[x + 1, y, temps + 1].avio == -1))
                {
                        if ((ect[x, y, temps + 1].avio == -1) || (ect[x + 1, y, temps].avio
== -1))
                        {
                                successors.Add(assignar_posicio(x + 1, y, temps
+ 1));
                        }
                        else
                        {
```

```
                                                    // Si no hi ha creuament
                                                    if(ect[x + 1, y, temps].avio != ect[x, y, temps +
1].avio)
                                                    {  // Si la posicio horitzontal seguent esta lliure )
{
                                                            successors.Add(assignar_posicio(x
+ 1, y, temps + 1));
                                                    }
                                            }
                                    }
                            }
                            if((y > 0) && (temps < maxTemps - 1))
                            { // Si no es la primera posicio de y
                                    if((ect[x, y - 1, temps + 1].avio == -1))
                                    {
                                            if ((ect[x, y, temps + 1].avio == -1) || (ect[x, y - 1, temps].avio
== -1))
                                            {
                                                    successors.Add(assignar_posicio(x,y - 1,temps +
1));
                                            }
                                            else
                                            {
                                                    // Si no hi ha creuament
                                                    if(ect[x, y - 1, temps].avio != ect[x, y, temps + 1].
avio)
                                                    {  // Si la posicio vertical anterior esta lliure ) {
                                                            successors.Add(assignar_posicio
(x, y - 1, temps + 1));
                                                    }
                                            }
                                    }
                            }
                            if((y < maxY - 1) && (temps < maxTemps - 1))
                            {  // Si no es la ultima posicio de y
                                    if((ect[x, y + 1, temps + 1].avio == -1))
                                    {
                                            if ((ect[x, y, temps + 1].avio == -1) || (ect[x, y + 1, temps].avio
== -1))
                                            {
                                                    successors.Add(assignar_posicio(x, y + 1, temps
```

```
+ 1));
                                                }
                                                else
                                                {
                                                        // Si no hi ha creuament
                                                        if(ect[x, y + 1, temps].avio != ect[x, y, temps +
1].avio)
                                                        {  // Si la posicio vertical seguent esta lliure ) {
                                                                successors.Add(assignar_posicio
(x, y + 1, temps + 1));
                                                        }
                                                }
                                        }
                                }
                                if(temps < maxTemps - 1)
                                {               // Si no es la ultima posicio de temps
                                        if(ect[x, y, temps + 1].avio == -1)
                                        {  // Si la posicio seguent esta lliure
                                                successors.Add(assignar_posicio(x, y, temps + 1));
                                        }
                                }
                        }

                        // Obte a partir de la posicio inicial
                        // el graf necessari del ECT i del calcul de sucessors per tal de poder
                        // inicialitzar el Min_Heap per a ser posteriorment tractat.
                        private void obtenirGrafIPrepararHeap(MinHeap heap, Stack pila, Posicio pos, int numAvio, ref int
numPendents)
                        {
                                pila.Push(pos);
                                Posicio posIni = pos;
                                while(pila.Count > 0)
                                {               // Mentre la pila no sigui buida
                                        Posicio posActual = (Posicio)pila.Pop();    // Obtencio del cim de la pila
                                        if(ect[posActual.x, posActual.y, posActual.z].tractat != numAvio)
                                        {
                                                NodeHeap elemHeap;
                                                elemHeap.pos = posActual;
                                                if ((posActual.x == posIni.x) && (posActual.y == posIni.y)
                                                        && (posActual.z == posIni.z))
                                                {
```

```
                                        elemHeap.salts = 0;
                                    }
                                    else
                                    {
                                        elemHeap.salts = maxTemps + 1;
                                    }
                                    heap.afegir(elemHeap);
                                    numPendents = numPendents + 1;
                                    ArrayList llistaFills = new ArrayList();
                                    calcularSuccessors(posActual, llistaFills);

                                    //std::vector<TPosicio>::const_iterator itFills;
                                    // Iteracio dels successors de la posicio actual, apilant els no
tractats
                                    foreach(Posicio fill in llistaFills)
                                    {
                                        if(ect[fill.x, fill.y, fill.z].tractat != numAvio)
                                        { // Si la posicio no s'ha tractat
                                            pila.Push(fill);
                                        }
                                    }
                                    ect[posActual.x, posActual.y, posActual.z].tractat = numAvio;
                                }
                            }
                        }

                        // Mètode que s'executa una vegada ha estat executat
                        // el metode d'obtencio de graf i inicialitzacio de heap, i que cerca en el heap
                        // els valors mes petits fins a trobar la posicio final, tot aplicant l'algorisme
                        // de Dijkstra.
                        private int cercarCamiMinim(MinHeap heap, int numPendents, Posicio posFi, int numAvio)
                        {
                            while(numPendents > 0) {
                                NodeHeap infoHeap = heap.obtenirMesPetit();
                                Posicio posHeap = infoHeap.pos;
                                int saltsActuals = infoHeap.salts;
                                heap.esborrar();
                                if((posHeap.x == posFi.x) && (posHeap.y == posFi.y) &&
                                    posHeap.z >= posFi.z) {
                                    return posHeap.z; // Es retorna el temps en el que arriba a la
posicio final.
```

```
                                        }
                                        else {
                                                ArrayList llistaFills = new ArrayList();
                                                calcularSuccessors(posHeap, llistaFills);

                                                // Iteracio dels successors de la posicio actual.
                                                foreach(Posicio fill in llistaFills) {
                                                if(ect[fill.x, fill.y, fill.z].pendent != numAvio) { // Si no s'ha
examninat previament.
                                                        Posicio posFill = assignar_posicio(fill.x, fill.y,
fill.z);
                                                        int saltsFill = heap.consultarValor(posFill);
                                                        if(saltsFill > (saltsActuals + 1)) {
                                                                // Es modifica el valor de aquells
en els que el valor
                                                                // del Min_Heap sigui major que
els salts acumulats del node + 1.
                                                                heap.canviarValor(posFill,
saltsActuals + 1);
                                                                // El successor fa guarda la
referencia dela posicio predecessora:
                                                                // el salt anterior en el cami de la
posicio inicial a la final.
                                                                ect[fill.x, fill.y, fill.z].nodeAnterior
= posHeap;
                                                        }
                                                        }
                                                }
                                                ect[posHeap.x, posHeap.y, posHeap.z].pendent = numAvio;
                                                numPendents = numPendents - 1;
                                        }
                                }
                                return -1; // En cas de no haver trobat un cami possible es retorna 1
                        }

                        // Assigna al ECT la sequencia de moviments
                        // calculada amb el Dijkstra per moure l'avio actual de l'inici fins al desti.
                        private void assignarCamiAect(Posicio inici, Posicio desti, int numAvio)
                        {
                                // Moviments de l'avio durant el ECT fins a situar-se en la seva posicio final
                                Posicio actual = desti;
```

```
                while((actual.x != inici.x) || (actual.y != inici.y) || (actual.z != inici.z))
                {
                        ect[actual.x, actual.y, actual.z].avio = numAvio;
                        actual = ect[actual.x, actual.y, actual.z].nodeAnterior;
                }
                ect[actual.x, actual.y, actual.z].avio = numAvio;
                // Es completa el ECT mantenint fixa la posicio de l'avio fins a MAX_TEMPS
                int numMoviment = desti.z + 1;
                while(numMoviment < maxTemps)
                {
                        // En cas de que un avio amb un nombre major de moviments que hagi estat
                        // previament planificat s'ubiqui en la posicio final de l'actual avio,
                        // es realitza anticipadament un moviment d'evasio per esquivar l'avio i
                        // posteriorment tornar a la posicio original de finalitzacio.
                        if(ect[desti.x, desti.y, numMoviment].avio != -1)
                        {
                                ArrayList llistaFills = new ArrayList();
                                desti.z = numMoviment - 1;
                                calcularSuccessors(desti, llistaFills);

                                Posicio posEvasio = (Posicio)llistaFills[0];
                                ect[posEvasio.x, posEvasio.y, posEvasio.z].avio = numAvio;
                        }
                        else
                        {
                                ect[desti.x, desti.y, numMoviment].avio = numAvio;
                        }
                        numMoviment++;
                }
        }

        private bool dijkstra(TipusCoordenada coordIni, TipusCoordenada coordFi, int numAvio)
        {
                Stack pila = new Stack();
                MinHeap heap;
                Posicio posIni, posFi;
                int numPendents = 0;
                int temps = 0;

                posIni.x = coordIni.x;
                posIni.y = coordIni.y;
```

```
                        posIni.z = temps;
                        posFi.x = coordFi.x;
                        posFi.y = coordFi.y;
                        posFi.z = calcularSaltsCamiMinim(coordIni, coordFi);
                        heap = new MinHeap(this.maxX, this.maxY, this.maxTemps);
                        obtenirGrafIPrepararHeap(heap, pila, posIni, numAvio, ref numPendents);
                        int numMoviments = cercarCamiMinim(heap, numPendents, posFi, numAvio);
                        if(numMoviments != -1)
                        {
                                posFi.z = numMoviments; // Unitat de temps en la que s'assoleix la posicio
final
                                assignarCamiAect(posIni, posFi, numAvio);
                                if(numMoviments > numMovimentsMajor)
                                {
                                        numMovimentsMajor = numMoviments;
                                }
                                return false;
                        }
                        else
                        {
                                return true;
                        }
                }
}
}
```

EventManager.cs

```
using System;
using SdlDotNet;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de EventManager.
        /// </summary>
        public class EventManager
        {
                public EventManager()
```

```
                {
                }

                public delegate void EntityMoveRequestEventHandler(object sender,
EntityMoveRequestEventArgs e);
                public event EntityMoveRequestEventHandler OnEntityMoveRequestEvent;

                public event QuitEventHandler OnQuitEvent;

                public void Publish(Object obj)
                {
                        if (obj.GetType().Name == "QuitEventArgs")
                        {
                                if (OnQuitEvent != null)
                                {
                                        //LogFile.WriteLine("EventManager has
received Quit event");
                                        OnQuitEvent(this, (SdlDotNet.QuitEventArgs)
obj);
                                }
                        }
                        else if (obj.GetType().Name == "EntityMoveRequestEventArgs")
                        {
                                OnEntityMoveRequestEvent(this,
(EntityMoveRequestEventArgs) obj);
                        }
                }

        }
}
```

InputController.cs

```
using System;
using SdlDotNet;

namespace PatrullaAguila
{
        /// <summary>
```

```
        /// Descripción breve de InputController.
        /// </summary>
        public class InputController
        {
                bool quitFlag;
                EventManager eventManager;

                public InputController(EventManager eventManager)
                {
                        this.eventManager = eventManager;
                        this.eventManager.OnQuitEvent += new QuitEventHandler(Subscribe);
                        Events.KeyboardDown += new KeyboardEventHandler
(this.KeyboardDown);
                        //Events.KeyboardUp += new KeyboardEventHandler(this.KeyboardUp);
                        Events.Quit += new QuitEventHandler(this.Quit);
                }

                private void Subscribe(object eventManager, QuitEventArgs e)
                {
                        //LogFile.WriteLine("InputController received a Quit event");
                        quitFlag = true;
                }

                private void KeyboardDown(object sender, KeyboardEventArgs e)
                {
                        if (e.Key == Key.Escape || e.Key == Key.Q)
                        {
                                eventManager.Publish(new QuitEventArgs());
                        }
                        else if (e.Key == Key.Keypad1)
                        {
                                eventManager.Publish(new EntityMoveRequestEventArgs
(TipusFormacio.Cunya));
                        }
                        else if (e.Key == Key.Keypad2)
                        {
                                eventManager.Publish(new EntityMoveRequestEventArgs
(TipusFormacio.Ascua));
                        }
                        else if (e.Key == Key.Keypad3)
                        {
```

```
                                        eventManager.Publish(new        EntityMoveRequestEventArgs
(TipusFormacio.Fletxa));
                                }
                                else if (e.Key == Key.Keypad4)
                                {
                                        eventManager.Publish(new        EntityMoveRequestEventArgs
(TipusFormacio.SuperMirlo));
                                }
                                else if (e.Key == Key.Keypad5)
                                {
                                        eventManager.Publish(new        EntityMoveRequestEventArgs
(TipusFormacio.Mirlo));
                                }
                                else if (e.Key == Key.Keypad6)
                                {
                                        eventManager.Publish(new        EntityMoveRequestEventArgs
(TipusFormacio.Poker));
                                }

                        }

                        private void Quit(object sender, QuitEventArgs e)
                        {
                                eventManager.Publish(new QuitEventArgs());
                        }

                        public void Run()
                        {
                                try
                                {
                                        while (!quitFlag)
                                        {
                                                while (Events.Poll())
                                                {
                                                        // handle events till the queue is
empty
                                                }

                                                try
                                                {
                                                        eventManager.Publish(new
```

```
TickEventArgs());
                                                                    }
                                                                    catch (SurfaceLostException)
                                                                    {
                                                                    }
                                                            }
                                                    }
                                                    catch
                                                    {
                                                            throw; // for this example we'll just throw it to the debugger
                                                    }
                                            }
                                    }
}
```

SDLViewer.cs

```
using System;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de SDLViewer.
        /// </summary>
        public class SDLViewer
        {
                EventManager eventManager;
                SDLView sdlView;

                public SDLViewer(EventManager eventManager)
                {
                        this.eventManager = eventManager;
                        sdlView = new SDLView(eventManager);
                        sdlView.CreateView();
                }

                public void ShowFormacio(TipusCoordenada[] coordenades)
                {
                        sdlView.ShowFormacio(coordenades);
```

```
                }
        }
}
```

SDLView.cs

```
using System;
using SdlDotNet;
using System.Drawing;

namespace PatrullaAguila
{
        public class SDLView
        {
                EventManager eventManager;
                int height;
                int width;
                int tamanyCella;
                int tamanyAvio;
                Surface surf;

                public SDLView (EventManager eventManager)
                {
                        this.width = 640;
                        this.height = 480;
                        this.tamanyCella = 50;
                        this.tamanyAvio = this.tamanyCella / 2;
                }

                public void CreateView()
                {
                        Video.SetVideoModeWindow(this.width, this.height, true);
                        this.surf = Video.Screen.CreateCompatibleSurface(width, height, true);
                        //fill the surface with black
                        this.surf.Fill(new Rectangle(new Point(0, 0), surf.Size), Color.Black);
                        Video.WindowCaption = "Patrulla Aguila";
                        Video.Mouse.ShowCursor(true);
                }

                public void ShowFormacio(TipusCoordenada[] coordenades)
```

```
{
        Video.Screen.Flip();
        this.surf.Fill(new Rectangle(new Point(0, 0), surf.Size), Color.Black);

        for(int i = 0; i < 7; i++)
        {
            for(int j = 0; j < 7; j++)
            {
                Box box = new Box((short)(j*tamanyCella), (short)(i*tamanyCella), (short)((j+1)*tamanyCella), (short)((i+1)*tamanyCella));

                this.surf.DrawBox(box, Color.White);
                Video.Screen.Blit(surf, new Rectangle(new Point(0, 0), Video.Screen.Size));
            }
        }

        for(int i = 0; i < coordenades.Length; i++)
        {
            Circle circle = new Circle((short)((coordenades[i].y * tamanyCella) + tamanyCella/2), (short)((coordenades[i].x* tamanyCella) + tamanyCella/2), (short)tamanyAvio);

            switch(i)
            {
                case 0:
                    this.surf.DrawFilledCircle(circle, Color.Red);
                    break;
                case 1:
                    this.surf.DrawFilledCircle(circle, Color.White);
                    break;
                case 2:
                    this.surf.DrawFilledCircle(circle, Color.Yellow);
                    break;
                case 3:
                    this.surf.DrawFilledCircle(circle, Color.Green);
                    break;
                case 4:
```

```
                                                this.surf.DrawFilledCircle(circle,
Color.Blue);
                                                break;
                                        case 5:
                                                this.surf.DrawFilledCircle(circle,
Color.Pink);
                                                break;
                                        case 6:
                                                this.surf.DrawFilledCircle(circle,
Color.Purple);
                                                break;
                                        case 7:
                                                this.surf.DrawFilledCircle(circle,
Color.Plum);
                                                break;
                                }
                                Video.Screen.Blit(surf, new Rectangle(new Point(0, 0),
Video.Screen.Size));
                        }
                        Video.Screen.Flip();

                }
        }
}
```

EntityMoveRequestEvent.cs

```
using System;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de EntityMoveRequestEventArgs.
        /// </summary>
        public class EntityMoveRequestEventArgs : EventArgs
        {
                TipusFormacio formacio;

                public EntityMoveRequestEventArgs(TipusFormacio formacio)
                {
```

```
                        this.formacio = formacio;
                }

                public TipusFormacio Formacio
                {
                        get
                        {
                                return this.formacio;
                        }
                }
        }
}
```

TickEventArgs.cs

```
using System;

namespace PatrullaAguila
{
        /// <summary>
        /// Descripción breve de TickEventArg.
        /// </summary>
        public class TickEventArgs : EventArgs
        {
                public TickEventArgs()
                {
                }
        }
}
```