Instituto Tecnológico de Costa Rica

Ingeniería en computadores

Segundo semestre 2024

Fecha: 24 de noviembre de 2024

Algoritmos y estructuras de datos 1

Grupo #2

Tercer proyecto;

Air War

Estudiantes:

Saddy Guzmán Rojas Carnet: 2023088184

Pablo Esteban Rivera Redondo Carnet: 2023395989

Contenido

Introducción: 3

Diseño: 4

Soluciones: 9

Diagramas 13

Checklist de user stories 15

Introducción:

El proyecto consiste en el desarrollo de un videojuego estilo Air war, utilizando los conocimientos obtenidos mediante el curso en cuanto a la implementación del paradigma de programación orientada a objetos, estructuras de datos, algoritmos de búsqueda y ordenamiento.

El juego consiste en controlar una nave que se mueve de izquierda a derecha de la pantalla, el objetivo es derribar a los aviones que se mueven por el mapa hacia diferentes destinos como, aeropuertos y porta aviones, el jugador tiene un tiempo limite y al finalizar se le mostrara los resultados del juego con la cantidad de aviones que logró destruir y sus ID ordenados mediante merge sort.

Diseño:

User stories

001- El objetivo del juego es destruir la mayor cantidad de aviones en un periodo de tiempo definido por el estudiante.

Yo como jugador quiero destruir la mayor cantidad de aviones en el tiempo limite de modo que sea divertido o que suponga un reto el intentar superar la marca.

Criterios de aceptación:

- Puedo controlar a mi jugador
- Puedo disparar con mi jugador
- Los aviones reaccionan a los disparos
- El juego se detiene en el tiempo limite y me mustra la cantidad de aviones destruidos

Casos alternativos:

- No se detectan colisiones
- No se construyen nuevos aviones
- El juego no se detiene con el tiempo

002- Generación aleatoria de aeropuertos, portaaviones y rutas. Se modela con un grafo utilizando listas de adyacencia.

Yo como jugador quiero que los aeropuetos, portaaviones y rutas se generen de manera aleatorea, de modo que cada partida sea completamente diferente.

Criterios de aceptación:

- Los nodos de agua y tierra se generan de manera aleatoria según un porcentaje definido.
- Los aeropuertos se generan en una posición aleatorea del grid y se verifica que este posicionado en un nodo de tierra.
- Los portaaviones se generan en una posición aleatorea del grid y se verifica que este posicionado en un nodo de agua.
- Los aviones al despegar escogen un destino de manera aleatoria y deciden cual es la mejor ruta mediante el grafo.

Casos alternativos:

- Tanto los aeropuertos como portaaviones deben aparecer en su tipo de nodo designado y la cantidad de estas estructuras se establece antes de comenzar el juego

- Si el algoritmo coloca más de una estructura en la misma posición, el sistema reintenta la asignación para evitar solapamientos. Se establece un mecanismo de validación previo a la asignación definitiva.

003-Batería antiaérea que se mueve en velocidad constante entre izquierda y derecha de la pantalla. El jugador presiona click para disparar balas con trayectoria recta y de velocidad variable según el tiempo que se presionó el click antes de lanzar la bala.

Yo como jugador quiero poder mover mi batería antiaérea de izquierda a derecha y disparar con una trayectoria recta y velocidad variable, de modo que pueda derribar los aviones de forma dinámica.

Criterios de aceptación:

- El jugador es capaz de moverse de izquierda a derecha de la pantalla
- El jugador puede variar la velocidad de la bala según el tiempo que mantenga el botón de disparo presionado.
- La bala tiene una trayectoria recta (hacia arriba) hasta colisionar con un avión o llegar al limite del mapa
- La salida del disparo siempre deberá ser el nodo actual de la batería antiaérea.

Casos alternativos:

- Cuando el jugador llegue al borde del mapa no podrá solo podrá moverse en la dirección opuesta
- La bala debe generarse hasta que el jugador suelte el botón
- La bala debe eliminarse si llega al límite del mapa
- Si la detección de la colisión falla la bala continuara su camino.

004- Entre aeropuertos y portaaviones, se generarán rutas aleatorias con distintos pesos

Yo como jugador quiero que las rutas entre aeropuertos y portaaviones tengan pesos aleatorios de modo que el costo y la estrategia de vuelo sean variables en cada juego.

Criterios de aceptación:

- Cada partida genera rutas que conectan aeropuertos y portaaviones de manera aleatoria, asegurando que: Todos los aeropuertos y portaaviones están conectados al menos a otro nodo (aeropuerto o portaaviones).
- Las rutas no deben ser redundantes (dos rutas iguales no existen en el mismo grafo).
- El peso de la ruta depende si se maneja por el agua o por tierra.

- Las rutas y el peso se representan visualmente en el grid como líneas que conectan nodos

Casos Alternativos:

- Verificación de que todos los nodos estén conectados.
- dos nodos están muy cerca para que sea imposible dispararles a los aviones, en este caso se implementa una distancia mínima que debe haber entre las estructuras.

005- Cuando un avión va a despegar, decide a qué destino quiere ir aleatoriamente y calcula la mejor ruta a ese destino, considerando los pesos anteriormente dados. Cuando un avión aterriza, dura una cantidad aleatoria de segundos antes de retomar una nueva ruta. Durante el tiempo de espera, el avión recarga una cantidad aleatoria de combustible.

Yo como jugador quiero que los aviones tengan un destino aleatorio y seleccionen la mejor ruta, además de que el avión recargue el combustible para que el juego sea más dinámico y duradero

Criterios de aceptación:

- El avión despega con un destino seleccionado de manera aleatoria.
- Para calcular las rutas se utiliza el algoritmo de **Dijkstra.**
- Durante el tiempo que el avión aterriza se recargara el combustible.
- De manera visual el avión debe seguir el movimiento de la ruta.

Casos alternativos

- El destino del avión no puede ser el mismo nodo del cual despego.
- Si el avión se queda sin combustible antes de llegar al destino será destruido
- El avión no encuentra una ruta valida para el destino seleccionado
- El portaaviones o aeropuerto no tiene suficiente combustible para recargar

007-Cada cierto tiempo los aeropuertos deben construir nuevos aviones. Por regulaciones internacionales, cada aeropuerto no podrá generar más aviones que la que permita sus hangares; por lo tanto, uno de los atributos de los aeropuertos es la cantidad de aviones que soporta su hangar. Los aviones deben tener un ID generado aleatoriamente utilizando GUIDs.

Yo como jugador quiero que los aeropuertos construyan nuevos aviones limitados por la capacidad de sus hangares, para garantizar que el flujo de aviones sea activo pero que también este limitado y haga el juego divertido.

Criterios de aceptación:

- El tiempo de construcción de los aviones estará definido
- Cada avión tiene un ID generado con guids
- Solo si el hangar tiene espacio disponible puede crear un avión
- El atributo de capacidad de hangar describe cuantos aviones puede crear/almacenar.
- Los aviones aparecerán en el aeropuerto o porta avión correspondiente

Casos alternativos

- Si el hangar alcanzo la capacidad máxima no creara más aviones

008- El sistema gestiona aviones autónomos operados por cuatro módulos de IA: Pilot (control activo), Copilot (respaldo en caso de falla), Maintenance (repara problemas físicos) y Space Awareness (monitorea sensores y radares). Cada módulo tiene un ID de tres letras aleatorias, un rol definido y registra sus horas de vuelo. Además, el sistema permite listar aviones derribados y ordenarlos por su ID utilizando Merge Sort.

Yo como creador del juego quiero que los aviones estén gestionados por cuatro módulos de IA, de modo que el juego tenga cierto dinamismo y complejidad.

Criterios de aceptación:

- Cada avión cuenta con los cuatro módulos.
- Los módulos cuentan con todos sus atributos, ID, Rol y horas de vuelo.
- Cada módulo gestiona el comportamiento del avión según su rol .
- Los aviones derribados generan una lista ordenada mediante merge sort de los IDs

Casos alternativos:

- Si el piloto falla, el copiloto asume el rol
- Si el copiloto falla, el avión se destruye

009- Para cada avión derribado, es posible obtener su tripulación (módulos Pilot, Copilot, Manteinance y Space Awarness) y ordenarlo por ID, rol o horas de vuelo, utilizando Selection Sort.

Yo como jugador quiero obtener la información de cada avión derribado de manera ordena, de modo que pueda revisar lo que logre jugando.

Casos de aceptación:

- Una vez termine la partida, el jugador podrá acceder a la información ordenada de los aviones
- La información debe incluir todos los atributos de los módulos
- El jugador puede ordenar la información por los atributos de cada módulo.

Casos alternativos:

- Atributos faltantes

010- En pantalla se debe mostrar todos los datos relevantes del juego. Por ejemplo, se deben mostrar los caminos calculados por los aviones, los pesos de cada ruta, los atributos de los aviones, entre otros.

Yo como jugador deseo ver todos los datos relevantes en la pantalla, de modo que estoy al tanto de lo que pasa en mi juego.

Casos de aceptación:

- Se visualizan los datos en la pantalla
- Los datos en la pantalla son correctos

Casos Alternativos:

- Los datos no se dibujan bien en la pantalla
- Los datos no son correctos

Soluciones:

- Creación del grid
  1. Solución La primera solución que se pensó para este proyecto y que de hecho se implementó hasta cierto punto fue utilizar la clase predefinida Panel, sin embargo, a pesar de sus ventajas como la facilidad en la forma de usarlo, no nos era conveniente del todo ya que no podíamos utilizarla de una manera más avanzada.

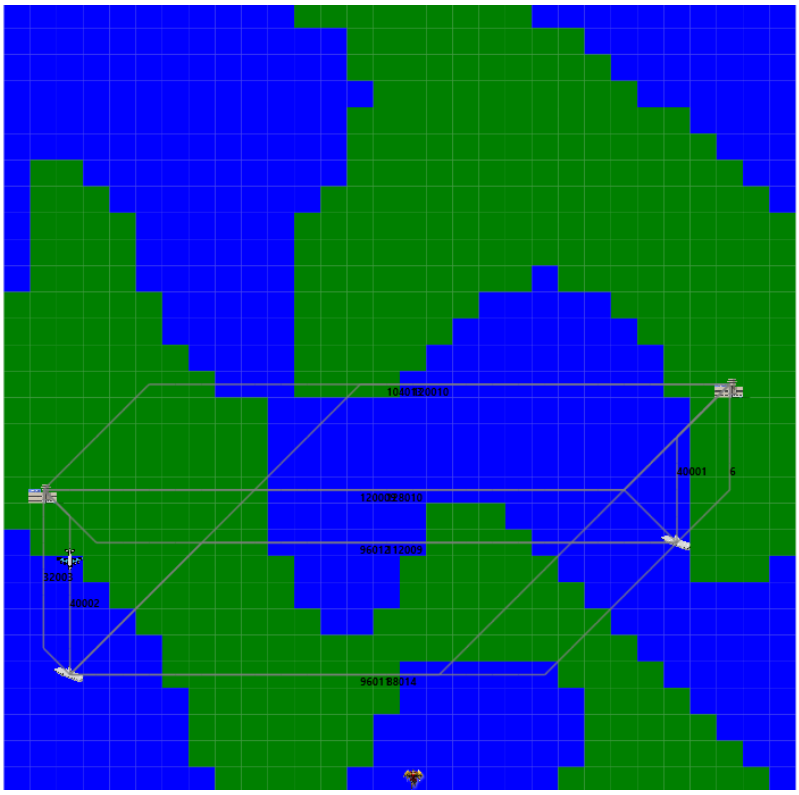
```
private void InicializarPaneles()
{
    int tamañoCelda = 30;

    for (int i = 0; i < tamaño; i++)
    {
        for (int j = 0; j < tamaño; j++)
        {
            Panel panel = new Panel
            {
                Size = new Size(tamañoCelda, tamañoCelda),
                Location = new Point(j * tamañoCelda, i * tamañoCelda),
                BackColor = Color.Blue // Color inicial para agua
            };

            paneles[i, j] = panel;
            coloresOriginales[i, j] = panel.BackColor; // Guardar el color original
            this.Controls.Add(panel);
```

  2. La segunda solución que fue la que se implemento de manera definitiva, fue crear el grid con nuestra propia clase nodo, la principal ventaja es que al ser una clase creada por nosotros podemos utilizarla de manera muy flexible, sin embargo, la desventaja es la complejidad comparada a usar simplemente una clase predefinida.

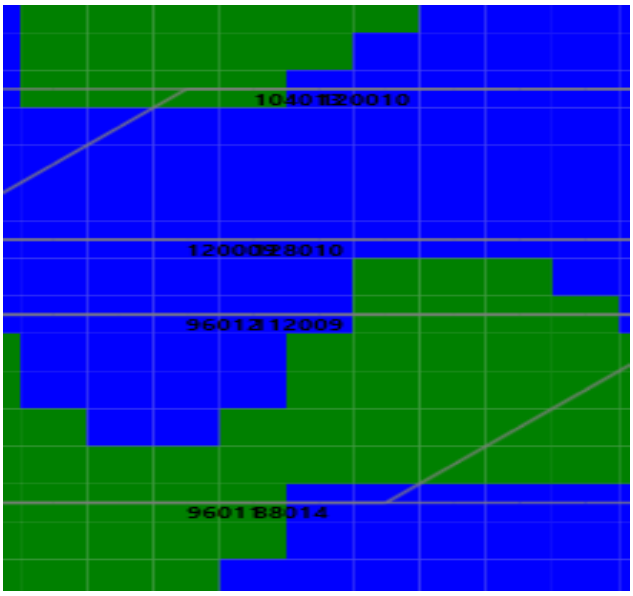
```
internal class Nodo //Se necesitan 8 referencias para c
{
    3 referencias
    public string Data { get; set; }
    7 referencias
    public Nodo Norte { get; set; }
    6 referencias
    public Nodo Sur { get; set; }
    8 referencias
    public Nodo Oeste { get; set; }
    8 referencias
    public Nodo Este { get; set; }
    5 referencias
    public Nodo Noroeste { get; set; }
    5 referencias
    public Nodo Noreste { get; set; }
    5 referencias
    public Nodo Suroeste { get; set; }
    5 referencias
    public Nodo Sureste { get; set; }
    9 referencias
    public TipoTerreno Terreno { get; set; } //En este
    8 referencias
    public bool TieneAvion { get; set; } //En este casc
    7 referencias
    public bool TieneAeropuerto { get; set; }
    7 referencias
    public bool TienePortaviones { get; set; }
    6 referencias
    public bool TieneJugador { get; set; }
```

- Diseño del juego
    1. Como primera opción al utilizar la clase predefinida pensábamos usar simplemente paneles de colores para representar los objetos del juego, los portaviones eran casillas negras, los aeropuertos grises, el jugador rojo y las balas amarillas, la principal ventaja de esto era la sencilles y rapidez a la hora de implementarlo, pero todo esto cambio cuando decidimos crear nuestros propios nodos para la matriz.

    2. Al utilizar nuestra propia clase nodo, teníamos la capacidad de colocar imágenes en las casillas, cosa que no era tan sencillo con la clase Panel predefinida, entonces esa es la principal ventaja, nuevamente la desventaja es que la creación de esta clase era mucho más compleja, pero obteníamos un mejor resultado.



- Dibujar las rutas en la pantalla
    1. Inicialmente pensábamos en utilizar un png con una bolita amarilla en cada casilla que marcaba la ruta en el mapa, sin embargo, era mas complicado hacer esto que nuestra opción definitiva.
    2. Utilizamos simplemente una función que dibujaba en el canvas una línea hacia cada ruta, la ventaja es que era mas sencillo que implementar la opción anterior en un proyecto casi concluido (esto fue lo ultimo que implementamos prácticamente), entonces la ventaja es la sencillez y la desventaja quizás sea un resultado un poco menos estético.
- Manejo de las rutas y los cálculos de peso
    1. Inicialmente se pensó en un algoritmo sencillo para calcular el peso de las rutas tomando únicamente en cuenta la distancia entre ellas, la ventaja es que sería una implementación demasiado sencilla pero no íbamos a obtener el resultado que realmente queríamos.
    2. De manera definitiva decidimos implementar un algoritmo que no solo tomara en cuenta la distancia, si no otros factores como el tipo de terreno por el cual pasa la ruta, es decir si pasa por tierra o agua, de eso dependerá el peso de la ruta, la ventaja es que tenemos un mejor resultado y mas completo, la desventaja es el tiempo que nos tomo desarrollarlo para que funcionara de manera adecuada.

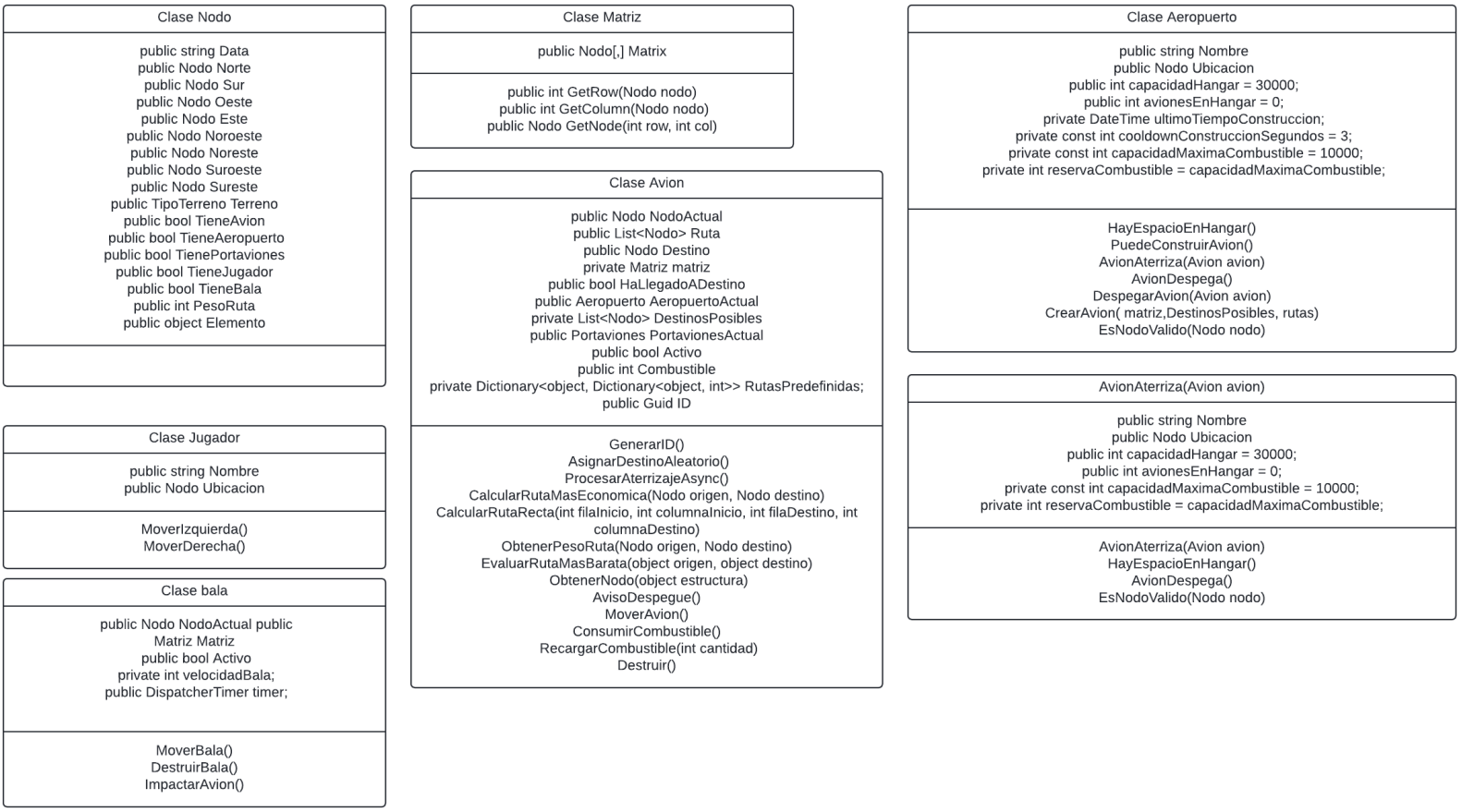


- Inconsistencias con las balas
    1. Una de las inconsistencias que tuvimos con respecto a las balas fue que al detectar una colisión, el elemento grafico de las balas no se eliminaban y se quedaban estáticas en una casilla, la primera solución que se nos ocurrió fue que crear una función encargada de la validación del elemento grafico de la bala, la ventaja es que al ser una función de validación nos iba a dar certeza de que se

estaba eliminando, la desventaja es que no estaba funcionando correctamente con el resto del código, así que no lo implementamos.

2. La solución que pusimos en practica para este problema fue la creación de una nueva lista llamada BalasAEliminar, en esta lista agregaríamos cada bala que detectaba una colisión con un avión, de esta manera nos asegurábamos de que todas las balas que colisionaban se iban a eliminar, además de llamar a la función InvalidateVisual(); que actualiza los elementos visuales de la pantalla, la ventaja de esta solución es que es tan sencilla que ni siquiera necesito de la creación de una nueva función, la desventaja es que por la ausencia de una función de verificación podrían pasar algunos errores.

Diagramas


Clase Nodo
public string Data
public Nodo Norte
public Nodo Sur
public Nodo Oeste
public Nodo Este
public Nodo Noroeste
public Nodo Noreste
public Nodo Suroeste
public Nodo Sureste
public TipoTerreno Terreno
public bool TieneAvion
public bool TieneAeropuerto
public bool TienePortaviones
public bool TieneJugador
public bool TieneBala
public int PesoRuta
public object Elemento
Clase Jugador
public string Nombre
public Nodo Ubicacion
MoverIzquierda()
MoverDerecha()
Clase bala
public Nodo NodoActual public
Matriz Matriz
public bool Activo
private int velocidadBala;
public DispatcherTimer timer;
MoverBala()
DestruirBala()
ImpactarAvion()
Clase Matriz
public Nodo[,] Matrix
public int GetRow(Nodo nodo)
public int GetColumn(Nodo nodo)
public Nodo GetNode(int row, int col)
Clase Avion
public Nodo NodoActual
public List<Nodo> Ruta
public Nodo Destino
private Matriz matriz
public bool HaLlegadoADestino
public Aeropuerto AeropuertoActual
private List<Nodo> DestinosPosibles
public Portaviones PortavionesActual
public bool Activo
public int Combustible
private Dictionary<object, Dictionary<object, int>> RutasPredefinidas;
public Guid ID
GenerarID()
AsignarDestinoAleatorio()
ProcesarAterrizajeAsync()
CalcularRutaMasEconomica(Nodo origen, Nodo destino)
CalcularRutaRecta(int filaInicio, int columnaInicio, int filaDestino, int columnaDestino)
ObtenerPesoRuta(Nodo origen, Nodo destino)
EvaluarRutaMasBarata(object origen, object destino)
ObtenerNodo(object estructura)
AvisoDespegue()
MoverAvion()
ConsumirCombustible()
RecargarCombustible(int cantidad)
Destruir()
Clase Aeropuerto
public string Nombre
public Nodo Ubicacion
public int capacidadHangar = 30000;
public int avionesEnHangar = 0;
private DateTime ultimoTiempoConstruccion;
private const int cooldownConstruccionSegundos = 3;
private const int capacidadMaximaCombustible = 10000;
private int reservaCombustible = capacidadMaximaCombustible;
HayEspacioEnHangar()
PuedeConstruirAvion()
AvionAterriza(Avion avion)
AvionDespega()
DespegarAvion(Avion avion)
CrearAvion(matriz,DestinosPosibles, rutas)
EsNodoValido(Nodo nodo)
AvionAterriza(Avion avion)
public string Nombre
public Nodo Ubicacion
public int capacidadHangar = 30000;
public int avionesEnHangar = 0;
private const int capacidadMaximaCombustible = 10000;
private int reservaCombustible = capacidadMaximaCombustible;
AvionAterriza(Avion avion)
HayEspacioEnHangar()
AvionDespega()
EsNodoValido(Nodo nodo)

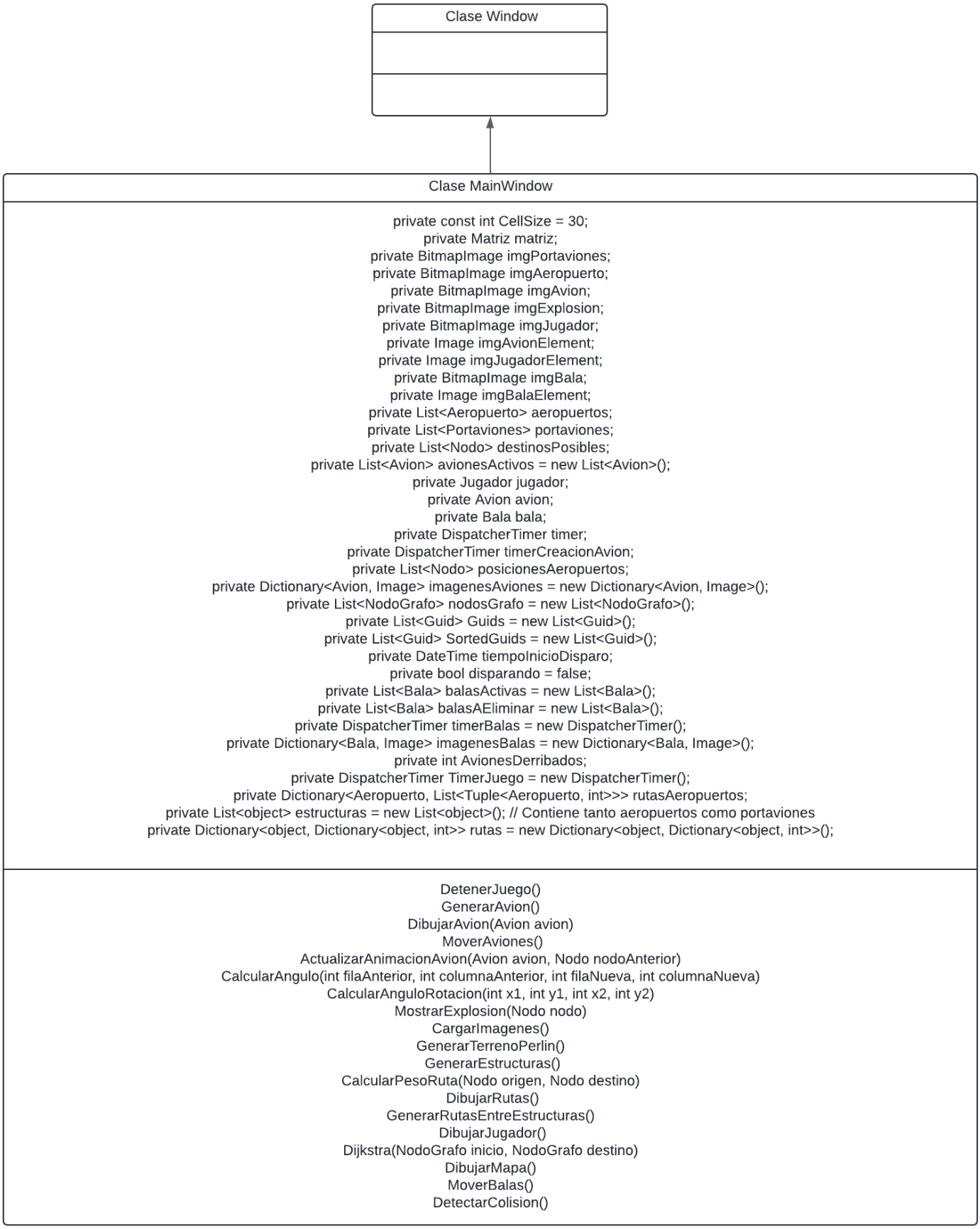
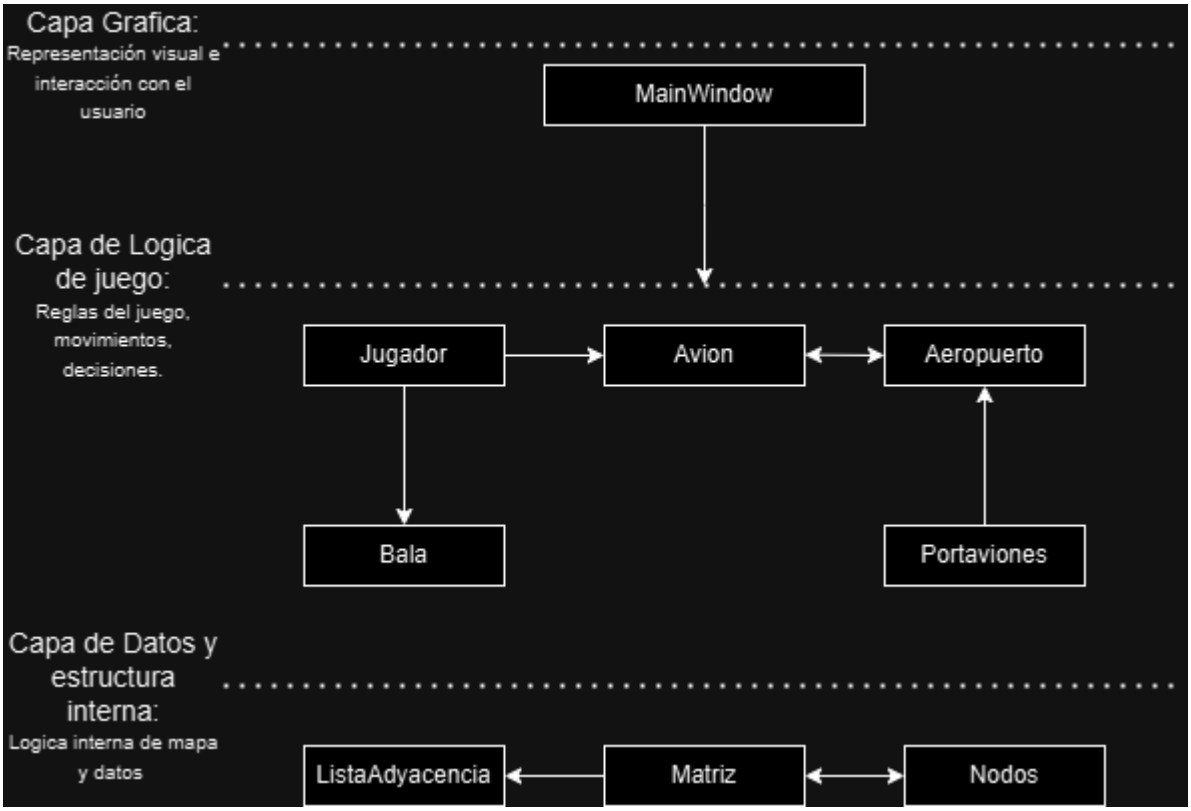

Clase Window
Clase MainWindow
private const int CellSize = 30;
private Matriz matriz;
private BitmapImage imgPortaviones;
private BitmapImage imgAeropuerto;
private BitmapImage imgAvion;
private BitmapImage imgExplosion;
private BitmapImage imgJugador;
private Image imgAvionElement;
private Image imgJugadorElement;
private BitmapImage imgBala;
private Image imgBalaElement;
private List<Aeropuerto> aeropuertos;
private List<Portaviones> portaviones;
private List<Nodo> destinosPosibles;
private List<Avion> avionesActivos = new List<Avion>();
private Jugador jugador;
private Avion avion;
private Bala bala;
private DispatcherTimer timer;
private DispatcherTimer timerCreacionAvion;
private List<Nodo> posicionesAeropuertos;
private Dictionary<Avion, Image> imagenesAviones = new Dictionary<Avion, Image>();
private List<NodoGrafo> nodosGrafo = new List<NodoGrafo>();
private List<Guid> Guids = new List<Guid>();
private List<Guid> SortedGuids = new List<Guid>();
private DateTime tiempoInicioDisparo;
private bool disparando = false;
private List<Bala> balasActivas = new List<Bala>();
private List<Bala> balasAEliminar = new List<Bala>();
private DispatcherTimer timerBalas = new DispatcherTimer();
private Dictionary<Bala, Image> imagenesBalas = new Dictionary<Bala, Image>();
private int AvionesDerribados;
private DispatcherTimer TimerJuego = new DispatcherTimer();
private Dictionary<Aeropuerto, List<Tuple<Aeropuerto, int>>> rutasAeropuertos;
private List<object> estructuras = new List<object>(); // Contiene tanto aeropuertos como portaviones
private Dictionary<object, Dictionary<object, int>> rutas = new Dictionary<object, Dictionary<object, int>>();
DetenerJuego()
GenerarAvion()
DibujarAvion(Avion avion)
MoverAviones()
ActualizarAnimacionAvion(Avion avion, Nodo nodoAnterior)
CalcularAngulo(int filaAnterior, int columnaAnterior, int filaNueva, int columnaNueva)
CalcularAnguloRotacion(int x1, int y1, int x2, int y2)
MostrarExplosion(Nodo nodo)
CargarImagenes()
GenerarTerrenoPerlin()
GenerarEstructuras()
CalcularPesoRuta(Nodo origen, Nodo destino)
DibujarRutas()
GenerarRutasEntreEstructuras()
DibujarJugador()
Dijkstra(NodoGrafo inicio, NodoGrafo destino)
DibujarMapa()
MoverBalas()
DetectarColision()

Diagrama de arquitectura:



Checklist de user stories

Las siguientes user strories fueron las que se lograron implementar de manera exitosa en el proyecto.

- ✓ 001
- ✓ 002
- ✓ 003
- ✓ 004
- ✓ 005
- ✓ 006
- ✓ 007
- ✓ 010

Por otro lado las que quedaron pendientes de implementación o su implementación fue parcial, por lo menos a la hora de redactar esta documentación fueron los user stories referentes a los requerimientos 008 y 009.