Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

MÉTODO DE LA INGENIERÍA

Fase 1. Identificación del problema.

Identificación de necesidades y síntomas:

- Las coordenadas enemigas deben desarrollarse a partir de algoritmos eficientes.
- Las tropas venusinas no cuentan con un programa que les permite estudiar las diferentes estrategias de batalla.
- La solución debe permitir generar las matrices con las filas y columnas deseadas para poder tener una mejor ubicación de las tropas enemigas.
- La solución al problema debe garantizar que se muestre las posiciones exactas de las tropas de Marte.

Definición del problema:

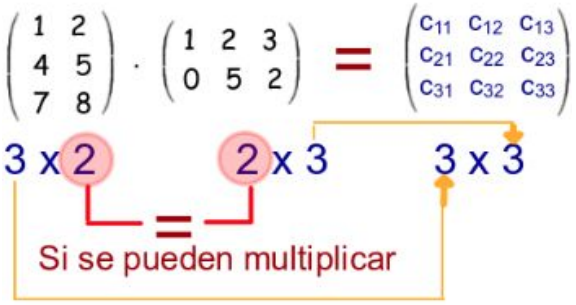
- Las tropas venusianas requiere un programa que permita descifrar las nuevas coordenadas exactas de las naves enemigas.

Fase 2. Recopilación de la información necesaria.

Definiciones:

Fuentes:
http://mimosa.pntic.mec.es/
https://www.matesfacil.com/matrices/

- *Multiplicación de matrices*
  Dadas dos matrices AA y BB de dimensiones m×n y n×p, respectivamente, se define su producto $A \cdot B = A \cdot B$ como la matriz de dimensión m×p tal que el elemento de la posición fila i y columna j es el resultado del producto de los vectores fila i de A y columna j de B. Para poder calcular el producto de matrices $A \cdot B$ se requiere el número de columnas de A sea el mismo que el número de filas de B.



- *Números primos*
  Un número primo es un número entero mayor que cero, que tiene exactamente dos divisores positivos. También podemos definirlo como aquel número entero positivo que no puede expresarse como producto de dos números enteros positivos más pequeños que él, o bien, como producto de dos enteros positivos de más de una

forma. Conviene observar que con cualquiera de las dos definiciones el 1 queda excluido del conjunto de los números primos.

- *Números enteros positivos*
  Cualquier entero mayor de cero.
  Por lo general, los enteros positivos se utilizan para representar cantidades enteras que se encuentran por arriba de un punto de referencia especificado.

Fase 3. Búsqueda de soluciones creativas.

- Dejar Matrices pre diseñadas para diferentes tipos de estrategias.
- Los valores aleatorios dentro de las casillas de las matrices sean entre 1 y 5.
- Los valores aleatorios dentro de las casillas de las matrices sean entre 1 y 3.
- Que todas las matrices sean de un tamaño nxn.
- Multiplicar las matrices combinando sus columnas.
- Realizar la multiplicación de las matrices componente a componente.
- Multiplicar las matrices combinando sus filas.
- Encontrar la matriz resultante como suma de matrices de rango 1.
- Al multiplicar las dos matrices realizarlo por cajas.
- Realizar diferentes variaciones internas al algoritmo que resuelve el problema de multiplicación.

Fase 4. Transición de las Ideas a los Diseños Preliminares

Alternativa 1. Multiplicar las matrices utilizando tres diferentes métodos de multiplicación.

- *Multiplicar las matrices combinando sus columnas.*
  Si A y B son compatibles para calcular C = AB, será porque $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times q}$. Las componentes de C se calculan mediante la ecuación siguiente:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, q.$$

- *Combinando columnas*.
  Construir un algoritmo que tiene como entrada una matriz A y un vector columna b y devuelve el vector columna c = Ab calculado como combinación lineal de las columnas de la matriz. Como comprobación adicional, comprobamos que efectivamente b sea un vector columna.

$$c = \sum_{k=1}^{n} a_{:,k} b_k$$

- *Combinando filas.*
  Construir un algoritmo que tiene como entrada un vector fila a y una matriz B y devuelve el vector fila c = aB calculado como combinación lineal de las filas de la matriz:

$$c = \sum_{k=1}^{n} a_k b_k$$

Alternativa 2. Multiplicar las matrices utilizando un único método de multiplicación haciéndole variaciones internas al algoritmo que lo resuelve.

Escoger entre las diferentes ideas de multiplicación propuestas en la lluvia de ideas, cuando se haya escogido uno se buscará un algoritmo base que lo resuelva y a partir de eso, empezar a desarrollar de diferentes formas el algoritmo, variando el procedimiento utilizado para multiplicarlas, como por ejemplo:

- Hacer el algoritmo recursivo.
- Cambiar la manera en la que se ejecutan los ciclos a fin de hacerlo más eficiente.

De esta manera, el programa deberá estar en capacidad de reconocer cuál de todos los algoritmos que se proponen para resolver la solución, deberá seleccionar para realizar la ejecución de la multiplicación de las matrices, teniendo en cuenta que la selección del algoritmo más eficiente será con base en el tamaño de la matriz.

Fase 5: Evaluación y selección de la mejor solución.

Criterios. A partir de las alternativas planteadas y la definición del problema, se plantean los siguientes criterios para escoger la solución más apropiada:

- **Criterio 1:** Facilidad de implementación. Se refiere a la posibilidad que hay de terminar la implementación a un plazo corto de tiempo. Esto puede ser:
  - [3] Improbable.
  - [2] Probable
  - [1] Muy probable.
- **Criterio 2:** Eficiencia. Se refiere al tiempo requerido por el algoritmo para terminar su ejecución. El cual puede ser:
  - [3] Alto.
  - [2] Aceptable.
  - [1] Bajo.

- **Criterio 3:** Consumo de memoria. Se refiere a la cantidad de almacenamiento que implementa el algoritmo para su debido funcionamiento. Puede ser:
    - [3] Baja.
    - [2] Normal.
    - [1] Excesiva.
- **Criterio 4:** Precisión de la solución. Se espera que los resultados obtenidos mediante la solución planteada puedan ser:
    - [2] Exactos.
    - [1] Inexactos.

Evaluación:

	Criterio 1	**Criterio 2**	**Criterio 3**	**Criterio 4**	**Total**
Alternativa 1. Multiplicar las matrices utilizando tres diferentes métodos de multiplicación .	Probable 2	Alta 3	Normal 2	Inexactos 1	8
Alternativa 2. Multiplicar las matrices utilizando un único método de multiplicación haciéndole variaciones internas al algoritmo que lo resuelve.	Muy probable 3	Aceptable 2	Normal 2	Exactos 2	9

Selección:
De acuerdo a los criterios planteados y la obtención de los resultados finales, se considera como mejor opción de diseño de solución la Alternativa 2. Teniendo en cuenta que es importante fortalecer los criterios en los que se presente mayor dificultad y, además, implementar la solución con las modificaciones que sean necesarias para lograr el objetivo final, manteniendo la esencia de esa Alternativa.

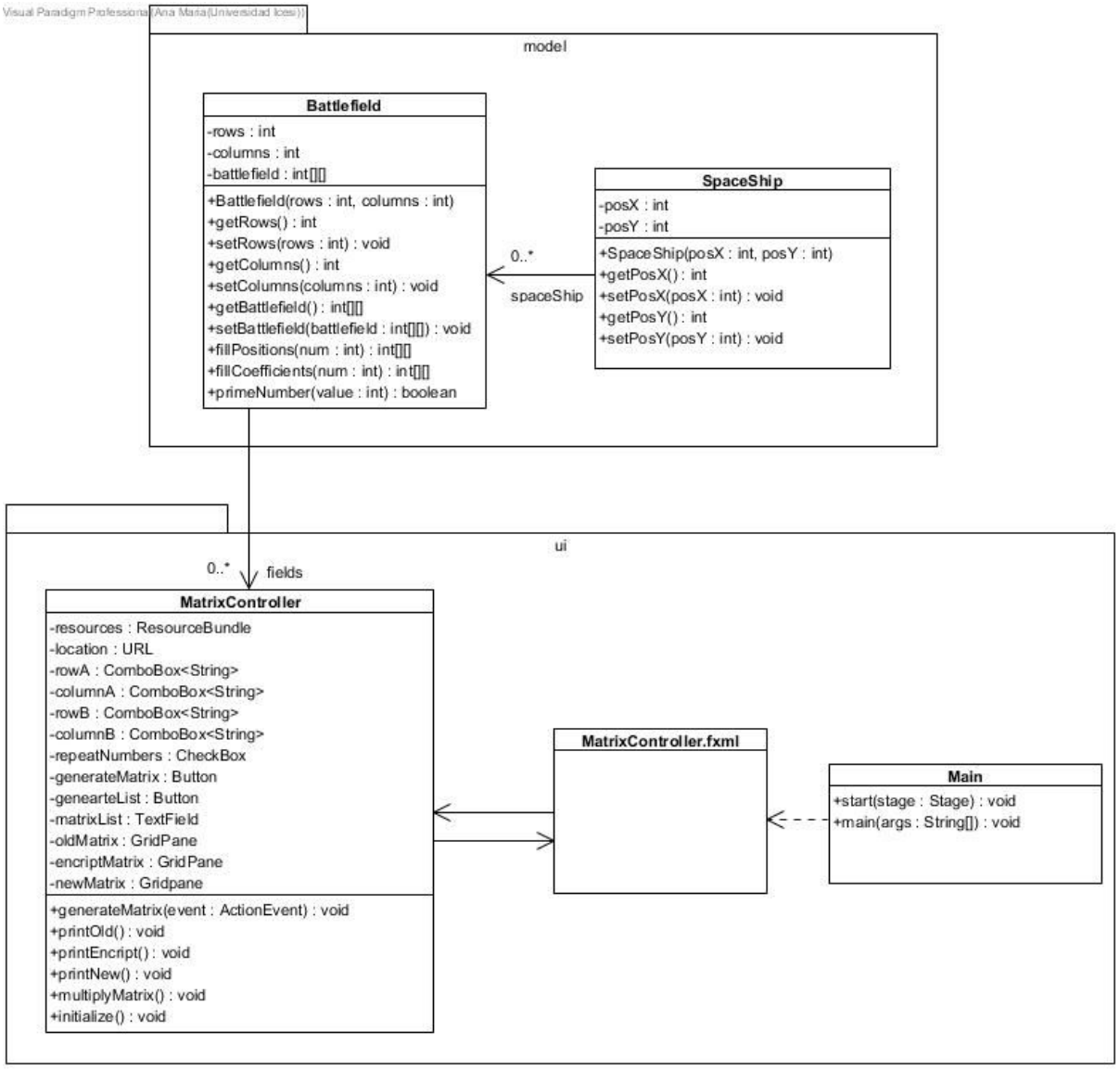
Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Fase 6. Preparación de informes y especificaciones.

Especificación del problema

- Problema: Generar una matriz a partir de la multiplicación de dos previas para hallar las posiciones de las naves de la tropa enemiga.
- Entradas: Las dos matrices a multiplicar, las cuales contienen los valores de las posiciones en todas las coordenadas x,y que las definen.
- Salidas: Una nueva matriz con las posiciones de las tropas enemigas.

Diagrama de clases



Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Diseños de casos de prueba

Configuración de los Escenarios

Nombre	Clase	Escenario
setupScenary1()	BattleFieldTest	Vacío
setupScenary2()	BattleFieldTest	battleF = new BattleField(5, 5)
setupScenary1()	SpaceShipTest	Vacío

Diseño de Casos de Prueba

Objetivo de la Prueba: Verificar la correcta creación de un BattleField				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Battle Field	BattleFi eld	setupScenary1	num = 4	Se ha creado correctamente un nuevo BattleField que inicializa a lo que contiene en su constructor.

Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Objetivo de la Prueba: Verificar que se genere correctamente la matriz con unas posiciones específicas.

Clase	Método	Escenario	Valores de Entrada	Resultado
Battle Field	fillPositions	setupScenary2	num = 3	Se ha generado correctamente el método que permite construir la matriz a partir de las entradas indicadas, en la que cada posición tiene un valor numérico específico entre 1 y 100.
Battle Field	fillCoefficients	setupScenary2	num = 5	Se ha generado correctamente el método que permite construir la matriz a partir de las entradas indicadas, en la que cada posición tiene un valor numérico específico entre 1 y 3.

Objetivo de la Prueba: Verificar la correcta verificación de un número primo.

Clase	Método	Escenario	Valores de Entrada	Resultado
Battle Field	BattleField	setupScenary2	testValue1 = 29 testValue2 = 6	Se ha verificado correctamente que el valor numérico ingresando por parámetro permite indicar la posición de una nave enemiga porque es un número primo.

Battle Field	BattleField	setupScenary2	testValue1 = 811 testValue2 = 10	Se ha verificado correctamente que el valor numérico ingresando por parámetro permite indicar la posición de una nave enemiga porque es un número primo.
Battle Field	BattleField	setupScenary2	testValue1 = 953 testValue2 = -10	Se ha verificado correctamente que el valor numérico ingresando por parámetro permite indicar la posición de una nave enemiga porque es un número primo.

Pseudocódigo algoritmos más relevantes

Algoritmo 1 - Generar matriz con las posiciones cada una con valor entre 1 y 100

- Fill positions

```
Positions = nxn
prime = 0
    for i = 0 a positions
        for j = 0 a positions en i
            value = random de 1 a 100
            positions en i, j = value
            if value is prime
                new spaceShip
                add spaceShip
                prime + 1
     if prime = 0
        counting = 0
        while counting < 5
            pos = random de 1 a num
            val = random de 1 a 100
            if val is prime
                positions en pos, pos = val
            counting + 1
return positions
```

Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Algoritmo 2: Definir si el valor que se encuentra en la posición es un primo

- Primes

```
contador = 2
prime = true
while prime and contador =/= value
        if value % contador = 0
              prime = false
        contador + 1
return prime
```

Algoritmo 3: Multiplicar las matrices para generar la matriz con las ubicaciones de las naves enemigas

- Multiply

```
rowA = rows of fields #1
colA = columns of fields #1
nxn A = nxn of fields #1
rowB = rows of fields #2
colB = columns of fields #2
nxn B = nxn of fields #2
if colA =/= rowB
        “error text”
matrixResultado = rowA x colB
multiplicacion = matrixResultado
for x = 0 a multiplicacion
      for y = 0 a multiplicacion en x
            for z = 0 a colA
                  multiplicacion en x,y = (A en x,z * B en z,y)
matrixResultado = multiplicacion
fields + MatrixResultado
```

Fill Positions

Positions = nxn	1
prime = 0	1
for i = 0 a positions	n+1
for j = 0 a positions en i	[n(n+1)/2] + 1
value = random de 1 a 100	n(n+1)/2
positions en i, j = value	n(n+1)/2

if value is prime	n(n+1)/2
new spaceShip	n(n+1)/2
add spaceShip	n(n+1)/2
prime + 1	n(n+1)/2
if prime = 0	1
counting = 0	1
while counting < 5	6
pos = random de 1 a num	5
val = random de 1 a 100	5
if val is prime	5
positions en pos, pos = val	5
counting + 1	5
return positions	1

Análisis complejidad temporal

Fill Coefficients

Positions = nxn	1
for i = 0 a positions	n+1
for j = 0 a positions en i	[n(n+1)/2] + 1
value = random de 1 a 3	n(n+1)/2
positions en i, j = value	n(n+1)/2
return positions	1

Primes

contador = 2	1
prime = true	1

while prime and contador =/= value	n - 1
if value % contador = 0	n - 2
prime = false	1
contador + 1	n - 2
return prime	1

Multiply

rowA = rows of fields #1	1
colA = columns of fields #1	1
nxn A = nxn of fields #1	1
rowB = rows of fields #2	1
colB = columns of fields #2	1
nxn B = nxn of fields #2	1
if colA =/= rowB	1
“error text”	1
matrixResultado = rowA x colB	1
multiplicacion = matrixResultado	1
for x = 0 a multiplicacion	n+1
for y = 0 a multiplicacion en x	[n(n+1)/2] + 1
for z = 0 a colA	n+1
multiplicacion en x,y = (A en x,z * B en z,y)	n
matrixResultado = multiplicacion	1
fields + MatrixResultado	1

Complejidad espacial

Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Fill Positions

	Variables	Cantidad
Entrada	num	n
	col	n
Salida	positions	n
Auxiliares	prime	1
	I	1
	J	1
	value	1
	sp	1
	counting	1
	pos	1
	posY	1
	val	1
		3n+9

Fill Coefficients

	Variables	**Cantidad**
Entrada	num	n
	col	n
Salida	positions	n
Auxiliar	value	1
		3n+1

Primes

	Variables	**Cantidad**

Alejandro Martínez
Cristian Lasso
José David Jiménez
Ana María Muñoz

Entrada	value	n
Salida	prime	n
Auxiliar	contador	1
		2n+1

Multiply

	Variables	**Cantidad**
Entrada		0
Salida		0
Auxiliar	fil_m1	1
	col_m1	1
	m1	n^2
	fil_m2	1
	col_m2	1
	m2	n^2
	matrixResultado	n^2
	multiplicacion	n^2
	x	1
	y	1
	z	1
		3n2 + 7