Daniel Alejandro Fernández Robles
A00354694
Camilo Enriquez Delgado
A00354532

Step 1. Identification of the problem.

The specific needs of the problematic situation, symptoms and conditions for resolution are listed below.

Identification of needs and symptoms:

> Venus needs a plan that helps them win the war against Mars for the territorial sovereignty of the 53 moons of Saturn and its rings.
>
> The Mars spacecraft are strategically located secretly and invisible to the troops of Venus.
>
> The algorithms for multiplying matrices and tracking the Martians must be efficient, so that they allow, very quickly, to make calculations on matrices of very large dimensions and find the right coordinates to attack Mars troops.
>
> To allocate the invisible Mars troops, Venus has some spacecraft locations from past battles and the possibility of knowing where the enemy will be located through a trusted predictor.

Definition of the problem:

> Venus requires a software module that allows them to track the Martians and helps them win the war and territorial sovereignty of the 53 moons of Saturn and its rings.

Requirements:

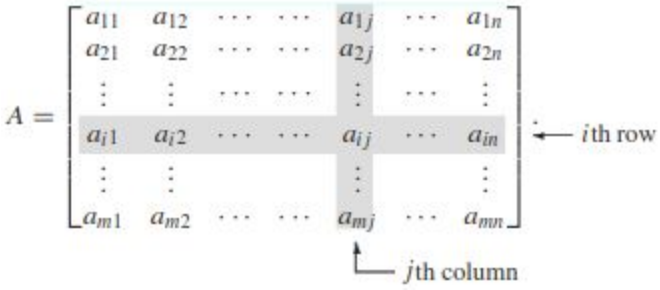
Name	**R.#1. Randomly generate a matrix**
Summarize	The program will allow you to randomly generate a matrix of considerably large dimensions and its values.
Input	● An integer giving the number of rows the matrix will have. ● An integer giving the number of columns the matrix will have. ● A boolean representing whether or not repeated elements are allowed.
Output	The requested randomly generated matrix with the specified restrictions.

Name	**R.#2. Multiply matrices**
Summarize	The program will allow you to compute the matrix product, using the method chosen by the user if the matrices are compatible with It(e.g: If the dimensions of the matrices are not 2^nx2^n where n $\in \mathbb{Z}^+$, then the divide and conquer and Strassen algorithms cannot be executed)
Input	• The matrix with Mars troops locations. • The secret matrix.
Output	The matrix resulting of multiplying the two matrices. This matrix contains the current locations of Mars troops.

Step 2. Gathering needed information.

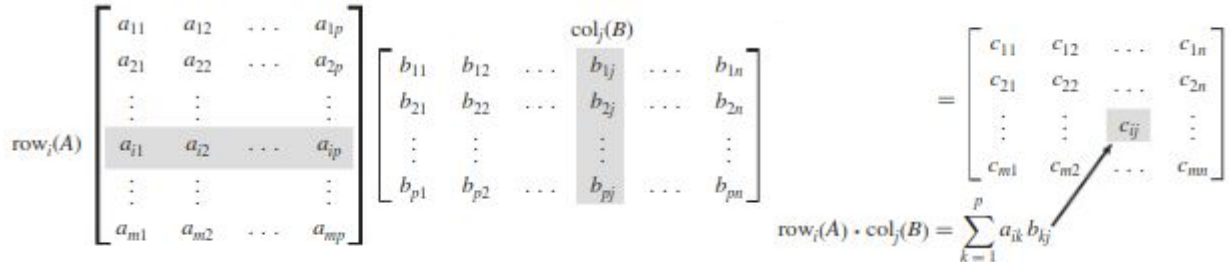
Matrix:

An m×n matrix A is a rectangular array of mn real (or complex) numbers arranged in m horizontal rows and n vertical columns.



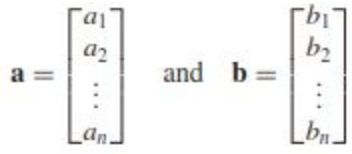
Matrix multiplication:

If A = $[a_{i,j}]$ is an m×p matrix and B = $[b_{i,j}]$ is a p×n matrix, then the product of A and B, denotes AB, is the m×n matrix C = $[c_{i,j}]$, in which the i, jth element is the dot product of the ith row, row $_i$(A), and the jth column, col $_j$(B), of B. The number of columns in matrix A must be equal to the number of rows in matrix B in order to multiply them.



Dot product:

The dot product or inner product of the n-vectors a and b is the sum of the products of corresponding entries. Thus, if



Then

$$\mathbf{a}\cdot\mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n = \sum_{i=1}^{n} a_ib_i$$

Step 3. Searching for creative solutions.

First option: Standard multiplication

The usual way to compute the matrix product is the one listed previously. The dot product between each row **i** in matrix **A** and column **j** in matrix **B** is allocated in row **i** and column **j** of the resulting matrix.

Second option: Divide and Conquer

This way for multiplying matrices is only applicable on squared matrices of dimensions $\mathbf{2^nx2^n}$ where **n** is a positive integer. It's focus is not to multiply the whole matrix but divide it into four submatrices and multiply them as if they were elements of a **4x4** matrix. The process is done recursively until the matrices that have to be multiplied are of dimension **1x1**. Then all the basic results are combined into the whole result.

Third option: Strassen algorithm

As the last one, this algorithm has a divide and conquer approach and also requires the two matrices to be multiplied to be squared matrices of dimensions $\mathbf{2^nx2^n}$ where **n** is a positive integer.

The Strassen algorithm splits the whole matrix into four submatrices($\mathbf{A_{1,1}}$ $\mathbf{A_{1,2}}$ $\mathbf{A_{2,1}}$ $\mathbf{A_{2,2}}$) and then defines seven new matrices:

- **P** = $(A_{1,1} + A_{2,2}) * (B_{1,1} + B_{2,2})$
- **Q** = $(A_{2,1} + A_{2,2}) * B_{1,1}$
- **R** = $A_{1,1} * (B_{1,2} - B_{2,2})$
- **S** = $A_{2,2} *(B_{2,1} - B_{1,1})$
- **T** = $(A_{1,1} + A_{1,2}) * B_{2,2}$
- **U** = $(A_{2,1} - A_{1,1}) * (B_{1,1} + B_{1,2})$
- **V** = $(A_{1,2} - A_{2,2}) * (B_{2,1} + B_{2,2})$

only using 7 multiplications (one for each submatrix) instead of 8 as the last divide and conquer option. We may now express $\mathbf{C_{i,j,}}$(The resulting matrix) in terms of each one of these submatrices:

- $\mathbf{C_{1,1}} = P + S - T + V$
- $\mathbf{C_{1,2}} = R + T$
- $\mathbf{C_{2,1}} = Q + S$
- $\mathbf{C_{2,2}} = P + R - Q + U$

We iterate this division process *n* times (recursively) until the submatrices degenerate into numbers (elements of the ring *R*). Then all the basic results are combined into the whole result.

Step 4. Stepping from ideation to preliminary designs (including modeling).

At the beginning of the solutions research, we found one that could work but because of the following reasons we didn't even list it:

Coppersmith-Winograd Algorithm:

-- There was not enough resources in order to get to understand what this solution did specifically.

-- Although It is more efficient than the other options, this algorithm implied a deep linear algebra knowledge which we are not acquainted.

The other alternatives were much easier to understand. Some of its characteristics are listed below:

Standard multiplication:

-- It's by far the easiest way to understand matrix product.

-- It's not just easy to understand but easy to implement.

-- Its time complexity is $O(n^3)$.

Divide and Conquer:

-- A divide and conquer algorithm works by recursively breaking down a problem into two or more subproblems of the same or related type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem.

-- Divide and conquer algorithms focus is to find more efficient ways to solve a problem.

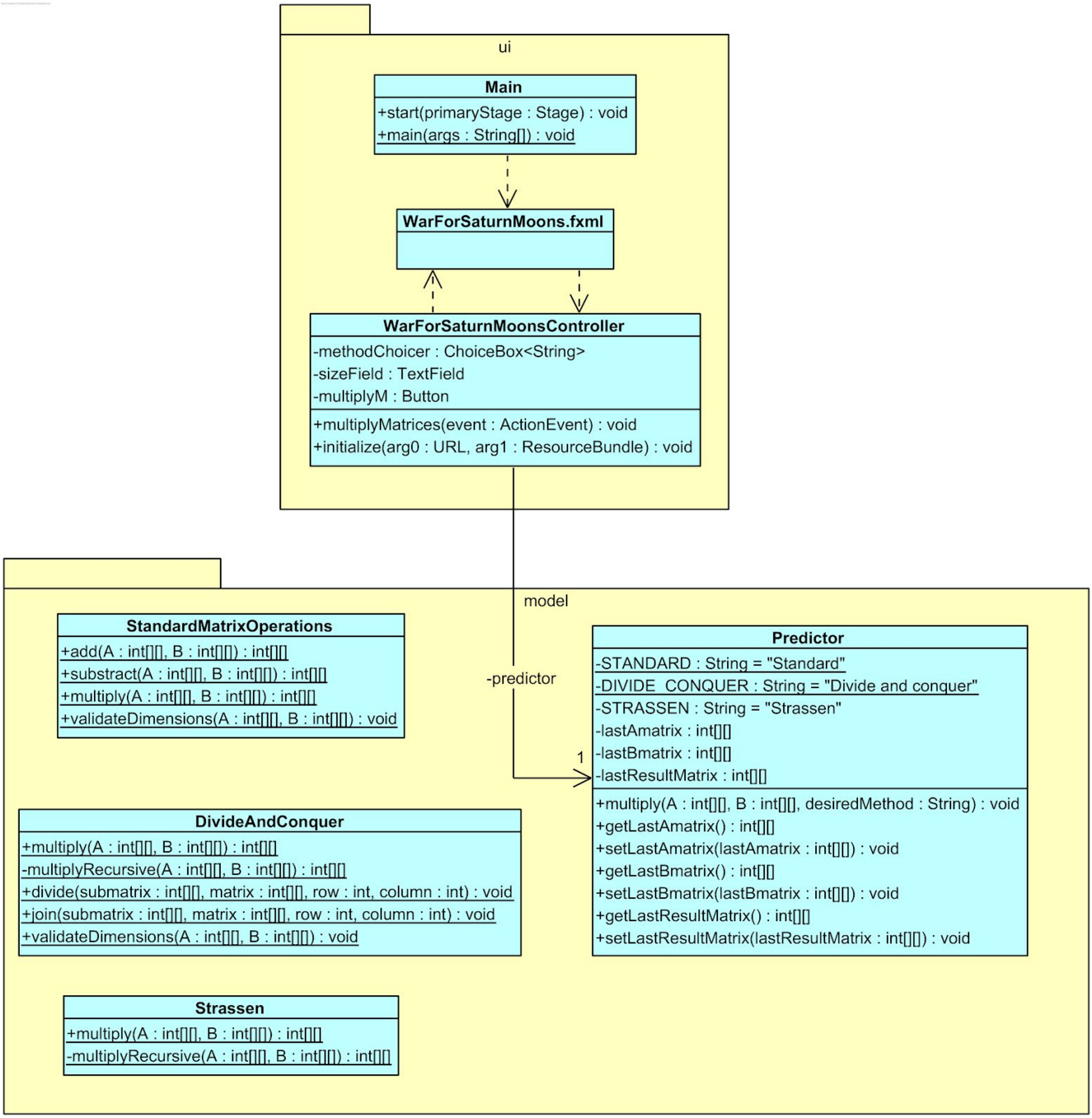
-- Its time complexity is $O(n^3)$.

Strassen algorithm:

-- It is a divide and conquer algorithm so it works by recursively breaking down the problem into two or more subproblems of the same or related type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem.

-- It reduces the number of matrix multiplications that should be done in each recursive step from 8 to 7.

-- Its time complexity is $O(n^{\log2(7)}) \approx O(n^{2.81})$.



Step 5. Evaluation and selection of preferred solution.

Criterion A: Solution Precision.

- [2] Accurate
- [1] Approximate

Criterion B: Efficiency.

- [6] Constant
- [5] Logarithmic
- [4] Linear
- [3] Quadratic
- [2] Something in the middle

[1] Cubic

Criterion C: Completeness.

[3] All solutions

[2] Some of them

[1] Just one or none

	Criterion A	Criterion B	Criterion C	Total
First Option	2	1	3	6
Second Option	2	1	2	5
Third Option	2	2	2	6

Selection:

In accordance with the previous table, the best options are the Standard multiplication and the Strassen Algorithm. Strassen should be applied whenever possible because of efficiency, but if It can't be applied then Standard multiplication is necessary.

Step 6. Preparation of reports, plans, and specifications.

Step 7. Implementation of the design.

Sources.

https://www.amazon.com/Introductory-Linear-Algebra-Applied-Course/dp/0131437402

https://en.wikipedia.org/

https://youtu.be/0oJyNmEbS4w