SPACE WARS SIMULATION

Project Report

2021-7-09

Made By:

1) Uttaran U Das (LCI2020047)

2) Avantika Modi (LCI2020026)

3) Preetish Patel (LCI2020011)

4) Hriday Gupta (LCI2020005)

TABLE OF CONTENTS

Introduction:

We have made a software simulation for a space-war. This simulation covers 4 elements, namely:

- StarShip
- MotherShip
- SpaceDebris and,
- Vacuum.

At the end of the simulation, we get all the statistics for major events during the lifecycle of the simulation and the final state of the war.

Project Model:

Traffic: The World is filled with each element occupying a certain amount of given space.

Life Cycle:

1. The War happens for a certain time period, ex: 10 or 1000000 units.
2. Every kind of ship will have a certain lifetime, after which it will corrode, which will constitute space debris. For example: when 10 StarShips corrode over time, they constitute one space unit of space debris.
3. After a certain amount of time, the space debris vanishes from the space.
4. When a certain spaceship is destroyed, it disappears from space.

Movement:

1. The SpaceShips can move in any random direction, i.e., Up, Down, Left, Right.
2. The SpaceShips cannot occupy spaces where there is space debris.

War Strategy:

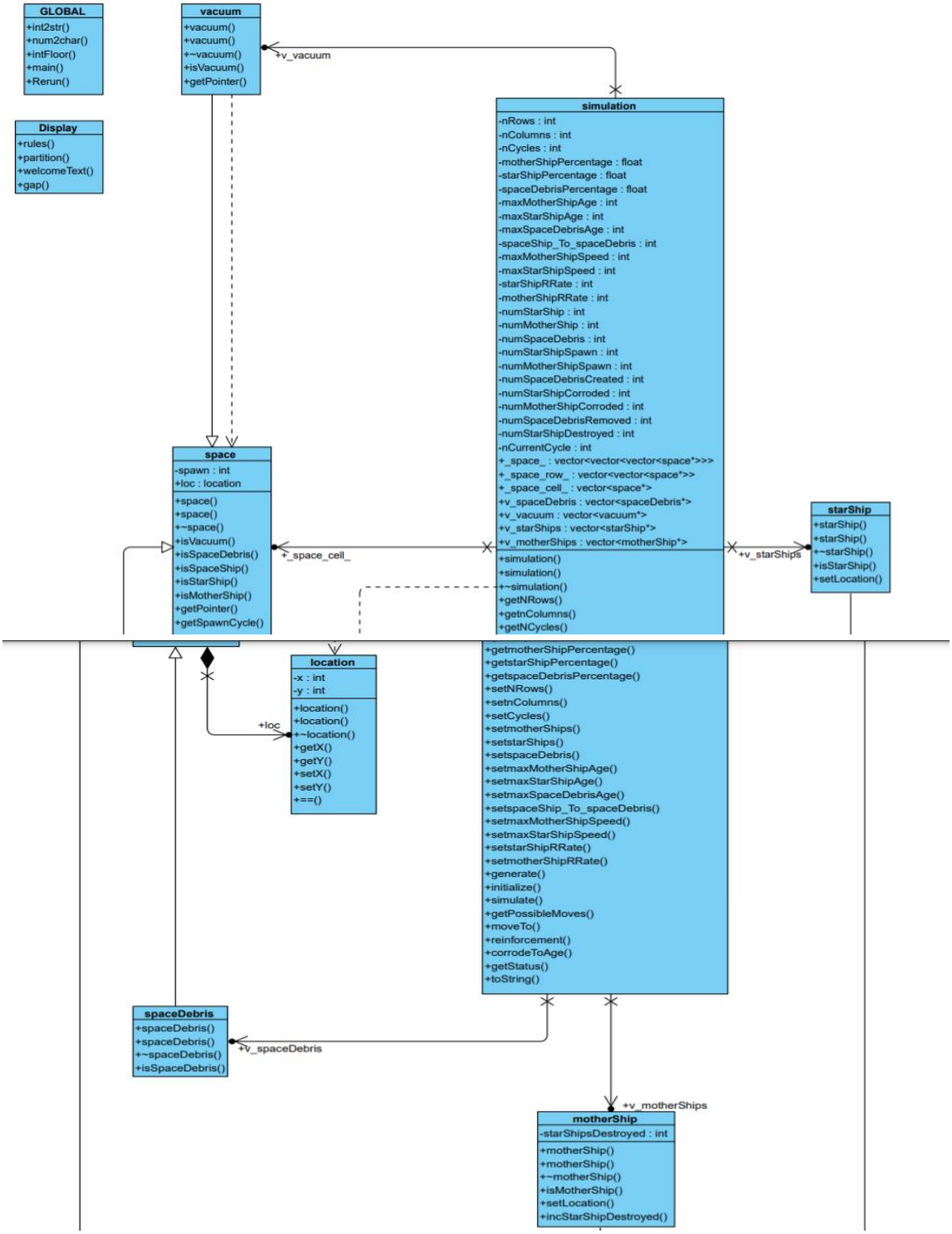
1. If a MotherShip and a StarShip arrive at the same position at the same time, then, MotherShip will destroy the StarShip, and the StarShip will disappear.
2. If two MotherShips arrive at the same position at the same time, they feel cornered and call reinforcements for help, which arrive at random positions in Space.
3. If two StarShips arrive at the same position at the same time, they call in reinforcements, which arrive at a random location in Space.

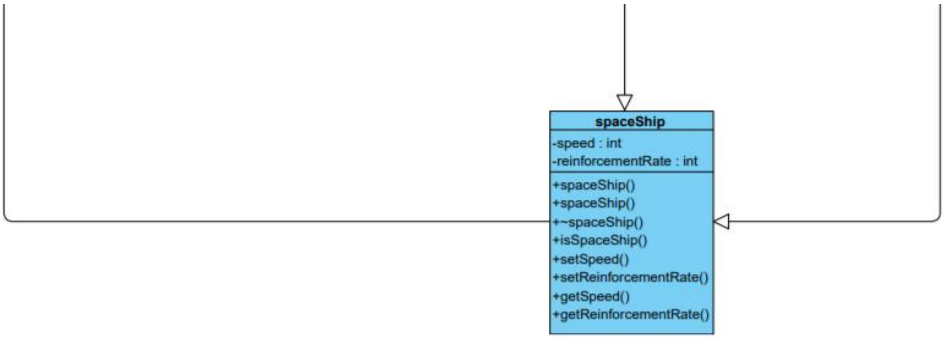
Results:

- StarShips at the end of the war.
- MotherShips at the end of the war.
- SpaceDebris at the end of the war.
- Who wins the war?

Class Diagram:

For better resolution refer to the github/zip folder.





Class Definitions:

1. ***space* Class:**

- ***space*** Class is defined as to be the superclass of three subclasses which are meant to represent the components of space. The three subclasses of ***space*** Class are:
    - ***spaceDebris***
    - ***spaceShip***
    - ***vacuum***
- It acts as an abstract class for the spacewar simulation.
- It contains a private variable namely ***spawn*** of integer data type that holds the start cycle for the object. It is initialized with the iteration number at which the object of *space* Class is created; defaults to 0.
- ***location loc*** shows the composition relationship between ***location*** class and ***space*** class. There is a strong life-cycle dependency between the two, as location holds the current location of the object, and if no component of space exists, there cannot be any location. Composition implies a relationship where the child cannot exist independent of the parent.
- Constructor overloading is implemented by having a default constructor and a parameterised constructor.
- It has a default *space* destructor.

- It has a getter function for the private ***spawn*** variable so that encapsulation is implemented.
- ***vacuum*** class depends on ***space*** class showing **dependency** relation between the two.

- It contains the following six **virtual** functions to be overridden by the derived classes :
    - **virtual bool isVacuum ()** which returns true when overridden by ***vacuum***.
    - **virtual bool isSpaceDebris ()** which returns true when overridden by ***spaceDebris***.
    - **virtual bool isSpaceShip ()** which returns true when overridden by ***spaceShip***.
    - **virtual bool isStarShip ()** which returns true when overridden by ***starShip***.
    - ***virtual bool isMotherShip ()*** which returns true when overridden by ***motherShip***.
    - ***virtual space *getPointer()*** which returns a pointer to the current object.
- It has an **association** relationship with class ***simulation***.
- *space* class is **non-navigable** from ***location*** class whereas ***location*** class is navigable from ***space*** class.

2. vacuum Class

- ***vacuum*** Class is defined as the subclass of the parent ***space*** class.
- It has a default constructor, and a parameterised constructor overloading.
- It has a default ***vacuum*** destructor.
- It has two methods:
    - **isVacuum()** being the overridden vacuum identifier method having the function of identifying vacuum.
    - **getPointer()** method which returns the pointer to the current vacuum object by using *this* keyword. This method is also the reason for the **dependency relationship** between ***space*** class and ***vacuum*** class.
- Public Inheritance is implemented while deriving from the superclass ***space***.
- It has an **association** relationship with class ***simulation***.

3. *spaceDebris* Class

- ***spaceDebris*** Class is defined as the subclass of parent ***space*** class.
- It has a default constructor, and a parameterised constructor overloading.
- It has a default ***spaceDebris*** destructor.
- It has a single boolean method:
    - ***isSpaceDebris()*** being the overridden ***spaceDebris*** identifier method having the function of identifying ***spaceDebris***.
- Public Inheritance is implemented while deriving from the superclass ***space***.
- It has an **association** relationship with class ***simulation***.

4. *spaceShip* Class:

- ***spaceShip*** Class defined as the subclass of parent ***space*** class and superclass of **starShip** and **motherShip** classes.
- ***spaceShip*** Class models common features of ***spaceships*** (***starships*** and ***motherships***).
- It has two private variables with integer datatype, which can only be accessed inside *spaceShip*:
    - ***speed*** storing the speed of the ***spaceShip*** in, the number of cells covered in a single cycle.
    - ***reinforcementRate*** storing the rate of reinforcement for the ***SpaceShip***
- It has a default constructor, and a parameterised constructor overloading.
- It has a default ***spaceShip*** destructor.
- It has five defined public methods:
    - ***isSpaceShip()*** being the overridden ***spaceShip*** identifier method having the function of identifying ***spaceShip***.
    - ***setSpeed()*** being a setter method for the ***speed*** attribute.
    - ***setReinforcementRate()*** being a setter method for ***reinforcementRate*** attribute.
    - ***getSpeed()*** being a getter method for the ***speed*** attribute.
    - ***getReinforcementRate()*** being a getter method for ***reinforcementRate*** attribute.

- It has a setter and getter methods for both ***speed*** as well as ***reinforcementRate*** private variables, so that encapsulation is implemented.
- Public Inheritance is implemented while deriving from the superclass ***space***.

5. *starShip* Class:

- ***starShip*** Class defined as the subclass for ***spaceShip***.
- ***starShip*** class has default constructor and parameterised constructor overloading.
- It has a default ***starShip*** destructor.
- It has two public methods ***isStarShip()*** and ***setLocation()***,.
    - ***isStarShip()*** being the overridden ***starShip*** identifier method having the function of identifying ***starShip***.
    - ***setLocation()*** being a setter method with the function of resetting the location of the ***starShip*** object.
- Public Inheritance is implemented while deriving from the superclass **spaceShip**.
- It has an **association** relationship with class ***simulation***.

6. *motherShip* Class:

- ***motherShip*** Class defined as the subclass of parent ***spaceShip*** class.
- It has a single private variable with integer datatype, i.e, ***starShipsDestroyed***, which stores the number of Star Ships destroyed by the Mother Ships.
- It has a default constructor, and a parameterised constructor overloading.
- It has a default ***motherShip*** destructor.
- It has three defined public methods:
    - ***isMotherShip()*** being the overridden ***motherShip*** identifier method having the function of identifying ***motherShip***.

- ***setLocation()*** being a setter method with the function of resetting the location of the ***motherShip*** object.
- ***incStarShipDestroyed()*** - the function to increment the ***starShipsDestroyed*** attribute.

- Public Inheritance is implemented while deriving from the superclass ***spaceShip***.
- It has an **association** relationship with class ***simulation***.

7. *location* Class

- ***location*** Class is designed to hold the location parameters of objects in the simulation, i.e, row number, column number.
- It has two private variables:
  - **x** represent row number
  - **y** represents column number
- It has a default constructor, and a parameterised constructor overloading.
- It has a default ***location*** destructor.
- It has the following methods:
  - ***getX*()** represents the getter function for x
  - ***getY*()** represents the getter function for y
  - ***setX*()** represents the setter function for x
  - ***setY*()** represents the setter function for y
- It also has two overloaded operators:
  - One returns the == comparison between two location elements.
  - The other returns the == comparison between two location pointer elements.
- There is a **dependency** relationship between ***location*** class and ***simulation*** class.
- There is a **composition** relationship between ***location*** class and ***space*** class.

8. *simulation* Class:

- The simulation class is implemented to handle the simulation.
- It contains the following private variables:
    - ***nRows*** of integer data type which represents the number of rows in the simulation field.
    - ***nColumns*** of integer data type which represents the number of columns in the simulation field.
    - ***nCycles*** of integer data type which represents the cycles/time the simulation has to run for.
    - ***motherShipPercentage*** of float data type which represents the percentage of initial cells for ***motherShips***.
    - **starShipPercentage** of float data type which represents the percentage of initial cells for ***starShips***.
    - **spaceDebrisPercentage** of float data type which represents the percentage of initial cells for ***spaceDebris***.
    - **maxMotherShipAge** of integer data type which represents the maximum age for ***motherShips***.
    - **maxStarShipAge** of integer data type which represents the maximum age for ***starShips***.
    - **maxSpaceDebrisAge** of integer data type which represents the maximum age for ***spaceDebris***.
    - **spaceShip_To_spaceDebris** of integer data type which represents the number of ***spaceShips*** to die before a ***spaceDebris*** is formed.
    - **maxMotherShipSpeed** of integer data type which represents the speed for ***motherShips***.
    - **maxStarShipSpeed** of integer data type which represents the speed for ***starShips***.
    - **starShipRRate** of integer data type which represents the reproduction rate for ***starShips***.
    - **motherShipRRate** of integer data type which represents the reproduction rate for ***motherShips***.
    - **numStarShip** of integer data type which represents the number of ***starShips***.
    - **numMotherShip** of integer data type which represents the number of ***motherShips***.
    - **numSpaceDebris** of integer data type which represents the number of ***spaceDebris***.

- **numStarShipSpawn** of integer data type which represents the number of ***starShips*** spawn.
- **numMotherShipSpawn** of integer data type which represents the number of ***motherShips*** spawn.
- **numSpaceDebrisCreated** of integer data type which represents the number of ***spaceDebris*** created.
- **numStarShipCorroded** of integer data type which represents the number of ***starShips*** that corroded during the simulation.
- **numMotherShipCorroded** of integer data type which represents the number of ***motherShips*** that corroded during the simulation.
- **numSpaceDebrisRemoved** of integer data type which represents the number of ***spaceDebris*** that were removed from the simulation due to crossing its maximum age.
- **numStarShipDestroyed** of integer data type which represents the number of ***starShips*** destroyed by ***motherShips***.
- **nCurrentCycle** of integer data type which represents the current iteration (used to set spawn on iteration for spaceShips; helps calculate age)

- It contains the following public attributes:
  - _***space***_ is a three-dimensional vector platform for the simulation to run on. 3rd dimension enables cases where more than 1 spaceShip falls on a single cell (2 ***starShips*** or 2 ***motherShips***). It stores *space* class objects and hence, is a reason for **association** relationship between ***simulation*** and **space** class.
  - _***space_row***_ is the row vector that is used to initialize the 3D space. It stores ***space*** class objects and hence, is a reason for **association** relationship between ***simulation*** and ***space*** class.
  - _***space_cell***_ is a single cell representing a single cell in 3D space. It stores *space* class objects and hence, is a reason for **association** relationship between ***simulation*** and ***space*** class.
  - ***v_spaceDebris*** is a spaceDebris vector to store all the ***spaceDebris*** objects. As it stores ***spaceDebris*** class objects, it is the reason for the **association** relationship between ***simulation*** and ***spaceDebris*** class.

- ***v_vacuum*** is a vacuum vector to store all the ***vacuum*** objects. As it stores ***vacuum*** class objects, it is the reason for the **association** relationship between ***simulation*** and ***vacuum*** class.
- ***v_starShips*** is a starShips vector to store all the ***starShip*** objects. As it stores ***starShip*** class objects, it is the reason for the **association** relationship between ***simulation*** and ***starShip*** class.
- ***v_motherShips*** is a motherShips vector to store all the ***motherShip*** objects. As it stores **mother*Ship*** class objects, it is the reason for the **association** relationship between ***simulation*** and ***motherShip*** class.

- Constructor overloading is implemented by having a default constructor and a parameterised constructor.
- It has a default ***space*** destructor.
- It contains the following methods:
  - ***getNRows()*** represents the getter function for ***nRows***
  - ***getnColumns()*** represents the getter function for ***nColumns***
  - ***getNCycles()*** represents the getter function for ***nCycles***
  - ***getmotherShips()*** represents the getter function for ***motherShipPercentage***
  - ***getstarShips()*** represents the getter function for ***starShipPercentage***
  - ***getspaceDebris()*** represents the getter function for ***spaceDebrisPercentage***
  - ***setNRows()*** represents setter function for ***nRows***
  - ***setnColumns()*** represents setter function for ***nColumns***
  - ***setCycles()*** represents setter function for ***nCycles***
  - ***setmotherShipPercentage()*** represents the setter function for ***motherShipPercentage***
  - ***setstarShipPercentage()*** represents the setter function for ***starShipPercentage***
  - ***setspaceDebrisPercentage()*** represents the setter function for ***spaceDebrisPercentage***
  - ***setmaxMotherShipAge()*** represents the setter function for ***maxMotherShipAge***
  - ***setmaxStarShipAge()*** represents the setter function for ***maxStarShipAge***
  - ***setmaxSpaceDebrisAge()*** represents the setter function for ***maxSpaceDebrisAge***

- ***setspaceShip_To_spaceDebris()*** represents the setter function for ***spaceShip_To_spaceDebris***
- ***setmaxMotherShipSpeed()*** represents the setter function for ***maxMotherShipSpeed***
- ***setmaxStarShipSpeed()*** represents the setter function for ***maxStarShipSpeed***
- ***setstarShipRRate()*** represents the setter function for ***starShipRRate***
- ***setmotherShipRRate()*** represents the setter function for ***motherShipRRate***
- ***generate()*** represents the function which generates the simulation field (the space of **nRows** X **nColumns** cells).
- ***initialize()*** represents the function which initializes the space with the required number of ***motherShips***, ***starShip*** and ***spaceDebris***.
- ***simulate()*** represents the function which starts and simulates the set environment for the set number of cycles.
- ***getPossibleMoves()*** represents the function which checks for possible moves the spaceShip can take from its current position.
- ***moveTo()*** passes *'dest'* an object of location as one of its arguments and represents the function which moves a spaceShip from current to destination. This method is also a reason for the **dependency** relationship between *location* class and *simulation* class.
- ***reinforcement()*** passes *'at'* an object of location as one of its arguments and represents the function which calls reinforcement for spaceShip at location *'at'* . This method is also a reason for the **dependency** relationship between ***location*** class and ***simulation*** class.
- ***corrodeToAge()*** represents the function which kills off ***spaceShips*** and ***spaceDebris*** based on the set max age parameters
- ***getStatus()*** returns string with current status of the simulation's environment
- ***toString()*** returns the results for a completed simulation * To be run after **simulate**()

- ***simulation*** class is **non-navigable** from the classes ***space***, ***spaceDebris***, ***vacuum***, ***starShip***, ***motherShip*** whereas these classes are **navigable** from ***simulation*** class.

Conclusion:

In this project, we were given a task to create using our Object-Oriented Programming skills. By using Microsoft C++ visual Code, we created a program to manifest a hypothetical **Space-War Simulation.**

Based on the project that has been completed, we had understood the use and importance of C++ programming in our daily life. Each software that had been created were composed by a complex C++ programming. In this project, we had identified the use of C++ programming in mathematical and logical Solving.

A conclusion we can probably draw from C++ programming is how to write a program efficiently is that there are no rigid rules to follow; everything is flexible. For example, small code size of a function is not necessarily good because a recursive function, while usually taking fewer lines of code, could have an incredibly big-time complexity as opposed to a non-recursive function.

This project helped us learn that how can we apply Object-Oriented Programming in our daily lives, and how to code efficiently and cleanly. By doing this project, we gained hindsight perspective, and we are sure that this will help us guide towards our respective futures.

Shortcomings:

- The simulation doesn't work with a 100% success rate. There are some cases where the simulation returns nothing.
- This is due to the incorporation of random moves of the respective objects, and the time for which the simulation runs, which produces error, or no output in some cases.
- The Simulation does not handle cases where a starShip could be surrounded by spaceDebris when the simulation is initialized.

The final simulation should look like this in its default state (*Values set within the code*).