OOP Teamwork Assignment

**Project Description**

**Background:**

Over the last decade, humanity has discovered thousands of exoplanets thanks to the Keppler and TESS missions. Some of these planets are with environmental conditions similar to those found on Earth. The James Webb Space Telescope will be launched in 2021. In a hypothetical scenario, the new telescope will find a brown dwarf star that will enter our Solar System in 80 years and push the Earth out of orbit. Our planet will become uninhabitable and die frozen as a rogue planet: a world untethered to any star, drifting through the void.

Design and implement **turn-based** **Space Evacuation Management Game (SEMG)** Console Application.

**Functional Requirements**

Application should support turns. Each move represents a calendar year and is called by "next", "turn" or another appropriate word at the discretion of the developers. Time is running in one direction only - forward. The game is divided into two periods. The first period is 80 moves between 2021 and 2100. During this period, ships can be developed and send to the nearby stars. After 2100, the Earth and the orbital infrastructure that produce ships will be destroyed. The second period lasts until year 2500. The goal of a player using the application is to achieve a maximum human population in more than one star system. For convenience, it is possible to introduce additional functionality for the next move (for example, "next 50" can skip 50 years ahead).

Application should support multiple **ships**. Each ship has **id,** **name**, **propulsion**, **speed**, **mass**, **destination**, **crew, population, annual probability for [critical failure](https://tvtropes.org/pmwiki/pmwiki.php/Main/CriticalFailure)**

* **ship id**
* id is generated by an automatic counter in and is in format (1, 2, …321…)
* should be unique in the application
* **ship name**
* should be unique in the application if not explicitly specified, the name is a combination of counter and propulsion (example: “000014 sail” or “000428 fusion engine”)
* **speed** is represented as a fraction of light speed (0.10 c is 10% of light speed)
* **mass** of the ship – in kg or metric **tons**. The weight of the ship is variable if the ship carries fuel (all propulsions except “sail”)
* **propulsion (**sail -speed 0.1c , fusion (nukes) speed 0.1c, fusion engine - speed 0.2c, matter anti matter speed 0.8c)
* **destination** - after setting the destination, the ship is launched in the next move.   
  Due to the high speeds and inertial forces, destination could not be changed
* **send** – check if the ship is Star Ship and send it to destination
* **crew**
* if unmanned mission the crew is only the ship AI
* if the mission is manned the crew is AI and people.

Application should support multiple planets. Each planet has Name, host Star Name, host Star Type, mass Earth M, radius Earth R, temp K, period Days, distance Light Years:

* Name
* Name is String between 5 and 50 symbols.
* Name should contain only letters, digits and “-“,
* host Star Name;
* Star Name is String between 5 and 50 symbols.
* Star Name should contain only letters, digits and “-“,
* host Star Type;
* mass – as fraction of Earth mass (between 0.5 and 5 is nice);
* radius as fraction of Earth radius (between 0.5 and 1.5 is nice);
* temp C;
* period in Days (how many days is a year on the planet)
* distance in Light Years from Earth;

Example:

| **Object** | **Star** | **Star type** | **Mass (M⊕)** | **Radius (R⊕)** | **Flux (F⊕)** | [**Teq**](https://en.wikipedia.org/wiki/Planetary_equilibrium_temperature)**(K)** | **Period (days)** | **Distance (ly)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Earth | Sun (Sol) | Yellow | 1.00 | 1.00 | 1.00 | 255 | 365.24 | 0 |
| Proxima Centauri b | Proxima Centauri | blue | 1.3 | 1.1 | 0.65 | 234 | 11.186 | 4.22 |

**Operations**

Application should support the following operations:

* Help – list all commands
* Create a new Space Object with fields Name, host Star Name, host Star Type, distance Light Years) the Space Object can be:
* Asteroid Field with additional fields has Metals, has Rare Earth Minerals
* Planet with additional fields mass in Earth M, radius in Earth R; temperature in C, period in Days, population
* Create a new ship with min fields ship Name, turns To Destination, destination. The ships can be:
* System Ships
* Star Ships. Only the Star Ships class (Colonial and Probe) have interstellar propulsion and can travel to other stars.
* Build Colonial Ship after set year (example: 2040).
* Find a way to limit capability for building ships for better EX (unlimited resources are boring)
* Send ship
* Refuse destination if you try to send Colony ship to Asteroid Field (Colony ship can travel only to Planets)
* At arrival the Colony Star Ship is lost and the crew become the population of the Planet
* At arrival convert Star Ship Probe to Miner (System Ship) if destination is Asteroid Field.
* At arrival convert Star Ship Probe to [Solar Power](https://www.google.com/search?rlz=1C1CHBD_bgBG823BG823&q=Solar+Power&spell=1&sa=X&ved=0ahUKEwiPuOuW0-XfAhVJhSwKHdBMCvsQkeECCCooAA) (System Ship) if destination is Planet.
* Lost ship command.
* Show all ships with destinations and expected year of arrival.
* Show all active ships.
* Show all Colony ships.
* Show all lost ships.
* Show all Space Objects (Planets and Asteroid fields)
* Show all Planets
* Support turn and multiple turns. (Example: next / next 35)
* Show data in the beginning of every turn about:
* arrived Colony ships at their destinations
* Current Year: 2041
* Years before / after Extinction Level Event
* Known Space Objects
* Colonized planets
* Total Population
* Total number of ships
* Active ships
* Lost ships
* Load / create Universe (create multiple space objects including Earth) for better UX – players can compere their results if they play on the same set of space objects.
* Extinction - If the load Universe command was used set the population of the Earth to 0 after Extinction Level Event.

**General Requirements**

* Follow the **OOP best practices**:
* Use data encapsulation
* Proper use inheritance and polymorphism
* Proper use interfaces and abstract classes
* Proper use static/final members
* Proper use enums
* Follow the principles of strong cohesion and loose coupling
* Use Streaming API
* Implement proper user input validation and display meaningful user messages
* Implement proper exception handling
* Cover functionality with unit tests (80% code coverage)
* Use Git to keep your source code and for team collaboration

Teamwork Requirements

Refer to the teamwork requirements document found along with the project requirements.