C++ Programming Coursework

1 Synopsis

It is the year 2317, and Earth is too crowded. To date, space travel has been limited to our own solar system, but a recent invention allows entire space fleets to be transported with near-light speed velocity. A number of Earth's biggest corporations have decided to grasp this opportunity and colonise the closest known habitable planet, Gaia, which is a mere 33 light-years away from earth. A race is gathering pace. To keep competition fair, the United Nations have specified a date on which all corporations will launch their fleets together, and have set up rules to determine who will eventually own Gaia.

The design of such a fleet is complex: corporations have access to a variety of spaceships, each with different characteristics. At the core of the fleet, there are the colonisation ships (CS), that carry colonists. Then there are the solar sail ships (SSS) which provide power converted from starlight to the fleet, medical ships that prevent disease, and military ships to protect from alien attacks.

Thanks to the recently invented faster-than-light (FTL) bubble ship, individual ships don't need their own FTL drive. There's a catch though: the heavier the fleet, the slower the bubble moves. The technology behind the FTL-bubble ship is kept secret by the United Nations Space Agency (UNSA), and UNSA is providing each corporation with a single FTL-bubble ship.

Under UN law - now universally accepted - war is illegal. To resolve rival corporations laying claim on the same planet, they have to abide to a simple set of rules: if a planet is uninhabited (i.e. the corporation is the first to arrive), the corporation is free to colonise it. If it is not empty, the arriving corporation can colonise it only if it arrives with more colonists than are living on the planet upon arrival. When this happens, the current inhabitants pack their bags and use the ships of the rival corp that just claimed their planet to start searching for another place to live. No colonists of the arriving corp are lost in this process. So, the more people in your fleet, the greater the chance you win the planet Gaia! There's a catch though: due to space limitations, a fleet's population can't grow in transit. On the other hand, a population on a planet grows by 5% year on year.

2 Coursework

It is your task to implement a space fleet that adheres by the rules specified in sections 1 and 3, implementing ships with specifications provided in section 4, and an interface to the game simulator specified in section 5. You are largely free to implement this however you like, as long as you adhere to the rules specified in this coursework.

A few constraints do apply: the program has to be written in iso-standard C++. You can only use standard library (STL) classes besides the classes you define yourself. The sourcecode should compile on our lab machine or at least you have to show us running it on your computer during the demonstration session in the lab. You have to hand in the coursework in a single zip file, with a filename that contains your group ID, via Moodle. See section 6 for details about the deliverables.

3 Game Rules

Below are the rules to which the game, and thus your code, must and will adhere:

- **Money** Each corporation (that's you, the student!) has the same fixed amount of money to spend on ships, to wit, 10,000.- UNP (United Nations Pounds). The price of ships is fixed and provided in Section 4.

- **Energy** Ships require energy. All ships in your fleet must have power, or else the entire fleet won't move.

- **Military** escort ships protect colony ships. How many colony ships each escort ship can protect depends on its size, and is listed in Section 4.

  It is your choice to protect your ships or not. Military escort ships prioritise protecting ships based on their population: bigger colony ships are protected before smaller ships.

- **Space Attack** Exactly halfway during the space voyage, the fleet is attacked by aliens. A random 25% of colony ships (rounded up) not protected by a military escort is destroyed, and the remains are left behind by the fleet.

- **Disease** Shortly after the attack by aliens, a disease breaks out on one of the remaining colony ships (randomly chosen). If the fleet does not have a medic ship, all colonists on that ship die. The ship (now empty) will continue to travel with the fleet though, and you should assume its weight is unmodified by the event.

- **Colonisation** As stated in the synopsis, what happens when a fleet arrives at Gaia is governed by two rules: if you are the first to arrive, your colonists settle on the planet and your colonisation population starts to grow (see below for details). If you arrive and the planet is already inhabited, two things can happen: either your fleet is carrying more colonists than the current planet population, and you take over, settling down with all your colonists, or if you arrive with fewer colonists in your fleet than there are people on the planet, you lose.

- **Planet population growth** Every year, the population of the planet grows by 5%. This happens instantly, after exactly one year has passed.

- **Distance to planet** The planet is 33 light years away from Earth.

- **Fleet speed** The speed of the FTL-Bubble is determined as follows:

  $$v = \frac{\alpha c}{\sqrt{w}} \qquad (1)$$

  where v is the fleet speed in meters per second, c is the speed of light, and w is the fleet weight, i.e. the sum of the weights of all ships in your fleet. α is set to 10 for this coursework.

- **Winner** The corporation (student) populating Gaia in the end has won.

4 The Ships

From the synopsis and rules sections, you probably gather that there are quite a few ships involved in this game. You will have to implement them all. Below is a list of all ships, per category. It also lists for every ship what the values are for their respective attributes. Note that you don't have to implement the Bubble Ship.

Colony Ships

There are three different colonisation ships, each with different costs, weight, and energy consumption. The properties of each ship are listed in table 1.

Name	Colonists	Cost	Weight	Energy Consumption
Ferry	100	500	10	5
Liner	250	1000	20	7
Cloud	750	2000	30	9

Table 1: Colony ship attributes.

Solar Sail Ships

There are two types of Solar Sail Ships, the standard ship Radiant, and the massive Ebulient. The Ebulient is so big, that it can often provide energy for an entire fleet. See table 2 for details.

Name	Energy Generation	Cost	Weight	Energy Consumption
Radiant	50	50	3	5
Ebulient	500	250	50	5

Table 2: Solar Sail Ship attributes.

Military Escort Ships

There are three different Military Escort Ship types: the light and fast Cruiser, the medium-sized Frigate, and the great alien thumper Destroyer. The ships have been designed to be fast and light, but are pretty expensive and consume a lot of power. Each ship will protect a number of colony ships (other ships won't get attacked). The second column in table 3 tells you how many, and the rest of the table provides all remaining information you need to implement these ships.

Each Military ship can protect one colony ship itself, regardless of the type of ship. In addition, Frigates and Destroyers carry a number of fighters. These can also be used to protect ships. Two fighters are necessary to protect one colony ship, regardless of the type of colony ship.

Name	Fighters	Cost	Weight	Energy Consumption
Cruiser	0	300	2	10
Frigate	10	1000	7	20
Destroyer	25	2000	19	30

Table 3: Military Escort Ship ship attributes.

Medic Ship

There is only one type of Medical ship, called the Medic. It is very small and efficient, but rather expensive. The benefit of this ship type is that if you choose to include one in your fleet, it will render your fleet immune to disease. Details of the ship are provided in table 4.

Name	Cost	Weight	Energy Consumption
Medic	1000	1	1

Table 4: Medical Ship attributes.

5 Interface with Game Environment

You must implement the following elements of the interface:

Listing 1: Interface

```
int Fleet::getWeight() const;                     // Returns cumulative
      weight of fleet
int Fleet::getEnergyConsumption() const;          // Returns cumulative
      energy consumption of fleet
int Fleet::getColonistCount() const;              // Returns cumulative
      colonist count of fleet
int Fleet::getCost() const;                       // Returns cumulative
      fleet cost
int Fleet::EnergyProduction() const;              // Returns cumulative
      energy production of fleet
int Fleet::countProtectedShips() const;           // Returns nr of
     colony ships protected in fleet
bool Fleet::hasMedic() const;                     // Returns True if
```

```
    the fleet has a medic ship, false otherwise
8 string Fleet::getCorporationName() const; // Returns         your
    chosen name of your corporation.
9 vector<Ship*> Fleet::protectedShips() const;// Returns a vector
     with ship numbers of protected colony ships
10 vector<Ship*> Fleet::unprotectedShips() const;              // Returns a
    vector with ship numbers of unprotected colony ships
11 vector<Ship*> Fleet::colonyShips() const;           // Returns a vector
     with ship numbers of all ships that are a colony ship
12 vector<Ship*> Fleet::shipList() const;              // Returns a vector
    with all ships in the fleet
13 void Fleet::destroyShip(Ship* i);                   // Removes ship i
    from the fleet
```

```
14 int Ship::getEnergyConsumption() const;        // Returns  energy
    consumption of a ship
15 int Ship::getWeight() const;                   // Returns  weight of
    a ship
16 int Ship::getCost() const;                     // Returns  cost of a
    ship
17 string Ship::getTypeName() const;              // Returns  the ship
    type, e.g. Ferry, Cruiser, or Ebulient. Note that spelling
    mistakes may effect your grade!
18 bool Ship::isDestroyed() const;                // Returns  true if
    the ship is destroyed, false otherwise
19 int ColonyShip::getColonistCount() const; // Returns       nr of
    colonists of a ship
20 void ColonyShip::infect();                     // Infects  a colony
    ship
21 bool ColonyShip::isInfected() const;           // Returns  True if
    the ship is infected with a disease, False otherwise
22 int SolarSailShip::getEnergyProduction const; // Returns energy
     production of Solar Sail Ship
23 int MilitaryEscortShip::getNrProtected() const;   //       Returns nr
     of colony ships protected by this ship
```

Besides the interface with the game environment, your code should im-plement the following functionality:

- A user interface that lets a user compose a fleet, and returns a pointer to that fleet. Call the function **Fleet* userInterfaceCreateFleet()**; The user interface should be interactive, and use the keyboard and screen only.

6 Deliverables

You need to hand in the following four files:

1. Fleet.h, containing the definition of your classes and functions.
2. Fleet.cpp, containing the implementation of your classes and functions.
3. groupID-fleet.dat, your fleet data file containing the fleet you want to enter in the competition.
4. groupID-report.pdf, which is your report containing class diagram, class diagram description, and class specification.

The format of the fleet data file should be as follows: Each line contains the information of the number of ships of each type, by putting the ship type name first, followed by whitespace, followed by the number of ships of that type. Note that if you don't have any ships of a particular type, you may simply omit it from your fleet data file. See Listing 2 for an example of a small fleet. N.B. DO NOT include the line numbering of the listing in your fleet data file!.

Listing 2: Fleet Data Example

```
1 Ferry   3
2 Cruiser 1
3 Medic   1
4 Radiant 1
```

The report should contain an image of the class diagram and the relation-ships between classes of your program. Make sure it includes all elements of the interface code you need to implement! With the class diagram should come a description of the diagram, in text. For each class in your class diagram, you will have to fill in a table just like that shown in Table 5. Note that, different to CRC-cards, we want you to use the actual function names and use the full signatures of the functions and variables (including const-ness).

Class name: Base class: Derived class: Component of: Component classes:
Function:

Variables	Description
Operations	Description

Table 5: Class specification template

Note that the Base class/Derived class fields should hold only the first generation up or down a class family tree. The Component of field should list classes that use this class as a component (usually as a member variable). The field Component classes on the other hand lists classes that are added as components or aggregates to this class. The Function: field should give a very short description of the function of this class (i.e. the reason for its existence). Variables and operations should have the same name as your variables and operations in the class diagram and in your C++ implementation, but you don't have to mention return types or whether they are public, private, or protected.

We will conduct a demo session in the Lab on 21st April 9:00am-11:00am.

Marking scheme:

1. **Documentation** 20%
   - Class diagram (INCLUDING RELATIONSHIPS) 5%
   - Class specification (using the template provided in Table 5) 10%
   - Description of Class diagram 5%
2. **Implementation** 70%
   - Compilation 15%
   - Adheres to Ship/Fleet specification 15%
   - Adheres to Interface specification 10%
   - Coding good practice 10%
   - Consistency with class diagram 10%
   - Code elegance 10%
3. **Oral exam** 10%

Submission deadline:

- 21st April 2016; 11:55pm (MYT)
- Late delivery of the report/demo will result in a penalty of 5% per day.