Design and code explanation

First, we have the main function in the Simulator.cpp file which basically is the entry point of the programme which represent the starting basic interface. It welcomes the user to an interface which welcomes them to ISRO and tell them something about ISRO. Which also tell the uses what type of mission that we offer in ISRO.

```
int main()
{
    ControlUnit *c1;
    cout << "                                    Welcome to INDIAN SPACE RESEARCH ORGANIZATION\n\n ";
    sleep(1);

    SpaceMission *mission;

    cout << "                                            Select Type of SPACE MISSION: \n"
         << endl;
    sleep(1);
    cout << "                                            1. Rover Launch Mission" << endl;
    sleep(1);
    cout << "                                            2. Satellite Launch Mission" << endl;       You, seconds ago •
    sleep(1);
    cout << "                                            3. Human Space Mission\n"
         << endl;
    sleep(1);

    cout << "                          🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀🚀\n";
    sleep(1);

    cout << "                          🚀🚀  1-->        TAKES A ROVER TO A PLANET            🚀🚀\n";
    sleep(1);
    cout << "                          🚀🚀  2-->     TAKES SATELLITE TO ORBIT OF EARTH       🚀🚀\n";
    sleep(1);
    cout << "                          🚀🚀  3-->     FOR INTER PLANATORY MAN MISSIONS        🚀🚀\n";
    sleep(1);

    cout << "                        Enter the type of mission You want to go on with ISRO...\n\n";
    cout << "                                  CHOICES any 1 of 3 missions (1/2/3) :: ";

    string input;
    cin >> input;
```

Then we take an input in the form of the string which has 3 options for different type Rocket missions. We have taken

```
ControlUnit *c1
SpaceMission *mission;
```

Class pointers. then to simulate we will simulate the space mission punter pointing to the given input and controlunit is also given as a parameter in each simulate to simulate different parts of the mission.

```
You, 3 hours ago | 2 authors (You and others)
class ControlUnit
{
public:
    void check_payload();
    void check_all_system_status();
    void iniatialising_launch_sequence();
    void mission_success();
};
```

```
void ControlUnit::check_payload()
{
    cout << "--------------------------------------\n";
    cout << "checking payload\n";
    cout << "payload checked successfully !\n";
}

void ControlUnit::check_all_system_status()
{
    cout << "--------------------------------------\n";
    printf("Final Checking all necessary systems ! \n");
    sleep(2);
    printf("Cryogenic engine check successful !\n");
    sleep(2);
    printf("Fuel check successful !\n");
    sleep(2);
    printf("Thermal screening check succesful!\n");
    sleep(2);
    printf("All System check successful !\n");
    sleep(2);       You, 6 days ago • New Commit added some new function to rocket class
    printf("Communication System check succesful !\n");
    sleep(2);
    printf("Mission is ready to go!\n");
    sleep(2);
    cout << "--------------------------------------\n";
}

void ControlUnit::iniatialising_launch_sequence()
{
    cout << "Initialing Launch Sequence\n";
    int i = 10;
    while (i >= 0)
    {
        cout << i << "\n";
        sleep(1);
        i--;
    }
    for (int i = 0; i < 3; i++)
    {
        cout << ".";
        sleep(1);
    }
    cout << endl;
}

void ControlUnit::mission_success()
{
    cout << "Congratulations mission is completed successfully!\n";
}
```

```cpp
class SpaceMission
{
private:
    string mission_name, purpose_of_mission;

public:
    SpaceMission();
    void about();
    string get_mission_name();
};
```

```cpp
SpaceMission::SpaceMission()
{
    cout << "Enter name of Space Mission : ";
    cin.ignore();
    getline(cin, mission_name);
    cout << "Enter purpose of Space Mission : ";
    getline(cin, purpose_of_mission);
    cout << "----------------------------------------\n";
    sleep(1);
    about();
}

void SpaceMission::about()
{
    sleep(1);
    cout << "The name of this SpaceMission is " << mission_name << endl;
    cout << "The purpose of this SpaceMission is : ";
    cout << purpose_of_mission << endl;
}

string SpaceMission::get_mission_name()
{
    return mission_name;
}
```

```
    while (input != "1" | input != "2" | input != "3")
    {
        if (input == "1")
        {
            mission = new Rover();
            simulate1(mission, c1);
            break;
        }
        else if (input == "2")
        {
            mission = new Satellite();
            simulate3(mission, c1);
            break;
        }
        else if (input == "3")
        {
            mission = new HumanSpaceMission();
            simulate2(mission, c1);
            break;
        }
        else
        {
            cout << "Press the correct key to prceed into ISRO Rocket launch mission : ";
            cin >> input;
        }
    }
}
```

So, this allows us to take input and if input is wrong, we take input again and again after that we simulate the selected mission by the user. First, we create a pointer of base class SpaceMission then we create the object of the derived or inherited classes like Rover or Satellite or Human exploration then first the object of the base class SpaceMission is created then its constructor is called then the objects of the further derived classes are created in order this is how inheritance works and it is very useful as it implements the hierarchical approach

1. Rover Launch Mission

```
void simulate1(SpaceMission *mission, ControlUnit *c1)
{
    ((Rover *)mission)->load_rover_in_rocket();
    ((Rover *)mission)->check_successful_loading();
    c1->check_payload();
    c1->check_all_system_status();
    c1->iniatialising_launch_sequence();
    c1->mission_success();
}
```

The function

```
((Rover *)mission)->load_rover_in_rocket();
((Rover *)mission)->check_successful_loading();
```

In these two is a function of Rover class which is inherited from SpaceMission class. And here we have made a SpaceMission type pointer. So we simply cannot use Rover class function by 'object.functionName()' method. To access that functions we have to use typecasting. In typecasting we specify that to which class this function belongs. The method of typecasting is: ((derived_class *)base_class_pointer)-> derived_class_function_name().
Same thing is applied to check_successful_loading () function.

```
void Rover::check_successful_loading()
{
    printf("performing rover loading checkes\n");
    sleep(2);
    printf("All checks of rover are successful\n");
}
void Rover::load_rover_in_rocket()
{
    sleep(2);
    printf("Rover is loading and is ready to launch in the orbit");
}       yash10019coder, 6 days ago • Major things are completed
```

2. Satellite Launch Mission

```
void simulate2(SpaceMission *mission, ControlUnit *c1)
{
    ((Satellite *)mission)->load_the_satellite();
    ((Satellite *)mission)->check_successful_loading();
    c1->check_payload();
    c1->check_all_system_status();
    c1->iniatialising_launch_sequence();
    c1->mission_success();
}
```

```
((Satellite *)mission)->load_the_satellite();
((Satellite *)mission)->check_successful_loading();
```

In these two is a function of Satellite class which is inherited from SpaceMission class. And here we have made a SpaceMission type pointer. So we simply cannot use Satellite class function by ‘object.functionName()’ method. To access that functions we have to use typecasting. In typecasting we specify that to which class this function belongs. The method of typecasting is: ((derived_class *)base_class_pointer)-> derived_class_function_name().
Same thing is applied to check_successful_loading() function.

```
void Satellite::check_successful_loading()        yash10019
{
    printf("performing satellite loading checkes\n");
    sleep(2);
    printf("All checks of satellite are successful\n");
}

void Satellite::load_the_satellite()
{
    sleep(2);
    printf("Satellite is loading in the rocket.");
}
```

3. Human Space Mission

```
void simulate3(SpaceMission *mission, ControlUnit *c1)
{
    c1->check_all_system_status();
    c1->iniatialising_launch_sequence();
    c1->mission_success();
}
```

```
class ControlUnit
{
public:
    void check_payload();
    void check_all_system_status();
    void iniatialising_launch_sequence();
    void mission_success();
};
```

We have some functon defined in ControleUnit class which we are using by aggrigation (a type of Association) for different missions.

```
void ControlUnit::check_payload()
{
    cout << "--------------------------------------\n";
    cout << "checking payload\n";
    cout << "payload checked successfully !\n";
}

void ControlUnit::check_all_system_status()
{
    cout << "--------------------------------------\n";
    printf("Final Checking all necessary systems ! \n");
    sleep(2);
    printf("Cryogenic engine check successful !\n");
    sleep(2);
    printf("Fuel check successful !\n");
    sleep(2);
    printf("Thermal screening check succesful!\n");
    sleep(2);
    printf("All System check successful !\n");
    sleep(2);       You, 6 days ago • New Commit added some new functi
    printf("Communication System check succesful !\n");
    sleep(2);
    printf("Mission is ready to go!\n");
    sleep(2);
    cout << "--------------------------------------\n";
}
```

```
void ControlUnit::iniatialising_launch_sequence()
{
    cout << "Initialing Launch Sequence\n";
    int i = 10;
    while (i >= 0)
    {
        cout << i << "\n";
        sleep(1);
        i--;
    }
    for (int i = 0; i < 3; i++)
    {
        cout << ".";
        sleep(1);
    }
    cout << endl;
}

void ControlUnit::mission_success()
{
    cout << "Congratulations mission is completed successfully!\n";
}
```