

Assessment Cover Sheet

This Assessment Cover Sheet is only to be attached to hard copy submission of assessments.



ASSESSMENT DETAILS

Unit title	Data Structures and Patterns	Tutorial /Lab Group	1	Office use only
Unit code	COS30008	Due date	30 Sep 2022	
Name of lecturer/tutor	Dr. Mark Tee Kit Tsun			
Assignment title	Problem Set 1			Faculty or school date stamp

STUDENT(S) DETAILS

	Student Name(s)	Student ID Number(s)
(1)	Alex Ngie Guan Ming	102765570
(2)		
(3)		
(4)		
(5)		

DECLARATION AND STATEMENT OF AUTHORSHIP

1. I/we have not impersonated, or allowed myself/ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is my/our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for me/us by any other person except where such collaboration has been authorised by the lecturer/tutor concerned.
4. I/we have not previously submitted this work for this or any other course/unit.
5. I/we give permission for my/our assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I/we understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I/we declare that I/we have read and understood the declaration and statement of authorship.

(1)	Alex	(4)	
(2)		(5)	
(3)		(6)	

Further information relating to the penalties for plagiarism, which range from a formal caution to expulsion from the University is contained on the Current Students website at <https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment>

Copies of this form can be downloaded from the Student Forms web page at <https://www.swinburne.edu.my/current-students/manage-course/exams-results-assessment/how-to-submit-work.php>

Contents

Task 1	3
Description	3
Concept	4
Task 2	5
Description	5
Concept	5
Implementation	7
Output	12
Troubleshooting	12
Task 3	13
Description	13
Concept	13
Implementation	13
Output	14
Troubleshooting	15
Task 4	16
Description	16
Concept	16
Implementation	16
Output	17
Troubleshooting	17
Task 5	18
Description	18
Implementation	18
Output	21
Troubleshooting	21

Task 1

Description

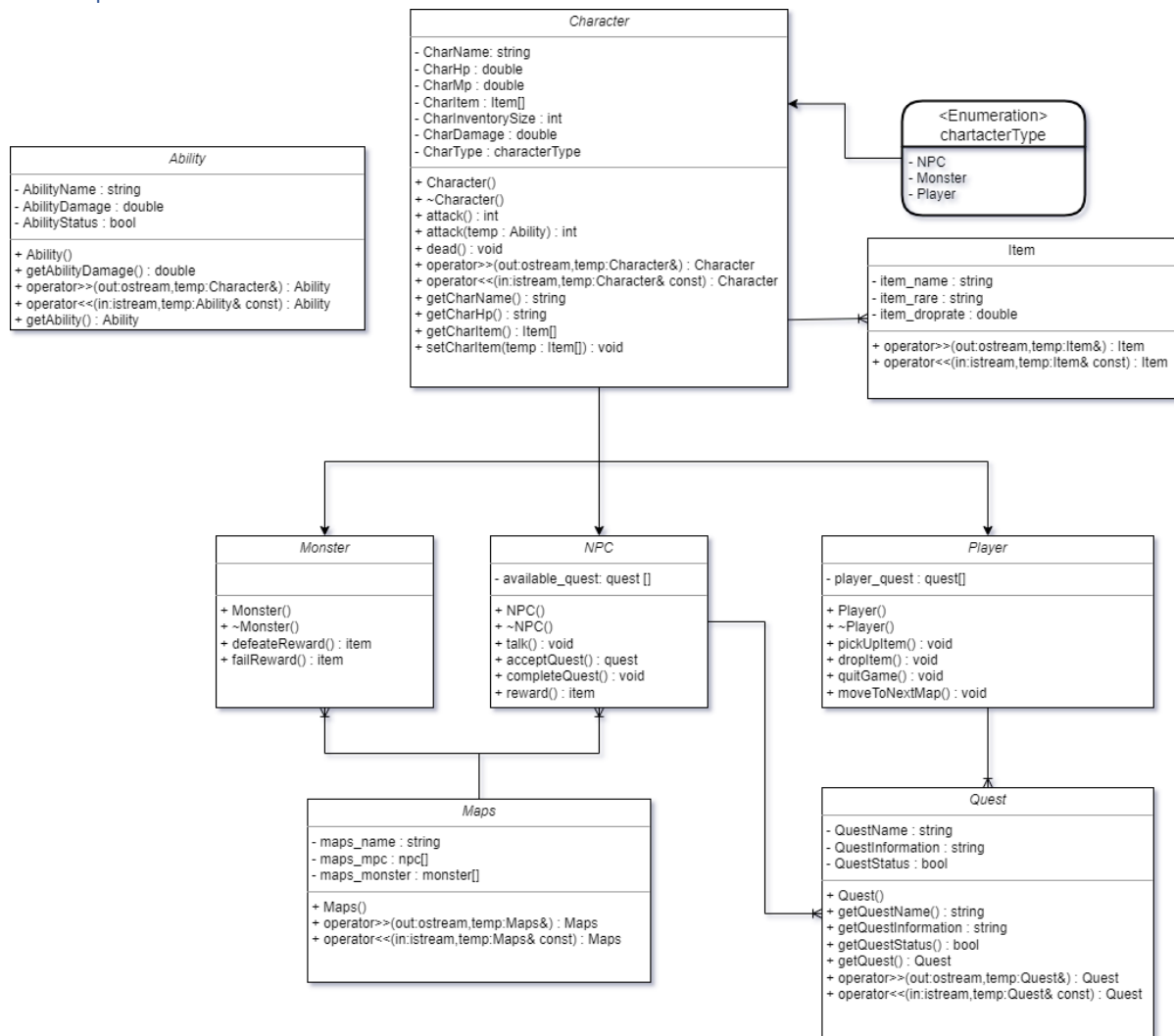
Game design and Structure

This is text-based single player action-adventure game. Player going to complete series of quest/mission to clear the game by using command to perform action like move, attack, collect item, talk with NPC etc. Depends on the command input by the player, it will perform the corresponding action that pre-designed in code. If any invalid command is input, error message will display and notify player. Player must follow the command designed. Example move (m – up, m – down, m – left, m - right), normal attack or skill attack (na – enemy1, sa - enemy2), collect item into inventory (c – item name, c – item name) etc. During the game progress, all enemy defeated, collected, or discarded items, completed quest etc, will storing in the memory.

In this game, player need move around different maps defeat the enemy to collect the material upgrade they weapon and equipment to increase damage and HP. Player have 2 ways to get the material, first is complete quest. Second is defeat the enemy, even failed to defeat enemy still have probability get the materials. The materials drop from the enemy will depends on the probability, more rarely materials have lower drop rate. Same enemies can be challenged repeatedly by the player. Maps and the skill ability will be unlocked based on the quest/mission completed by player. During the battle mode, player allowed to select attack enemy they challenge by normal attack or skill attack. But enemy attack pattern is randomly, depends on the code. if player HP reaches zero before the enemy, the challenge is considered a failure, else victory.

Once player defeat last boss, the game is completed and clear.

Concept



Polymorphism is fit with the design of character. In this game included various of character that's enemy, player, and NPC. Three of them consist of same information such as HP, skills, name, energy, current position, level, id of inventory etc. Between this, function overloading is used on the attack action. Attack action perform by the user fully depends on the user command input, but enemy is based on the probability, so there are two functions of attack, with and without parameter. By using the Polymorphism, it able to reduce code redundancy.

Queue is fit with the design of maps status, locked, and unlocked. Following game design, user must defeat previous maps enemy or complete the quest, after that next maps will be unlocked. Once player was done, just remove the first item in the queue and the second queue will be the first one. By using the Queue, the first in first out rule able to achieve this function easily without assign any ID or remark the priority of each quest or maps. It reduces a lot of calculate and memory usage of computer.

Task 2

Description

This task presenting how class derive from another class by inheritance. Public and protected properties or method will inherit from parent class to child class. Between this, child class also can have it own properties and methods to perform it own action. Below diagram showing entities derived 3 different class that's zealot, dragoon, and carrier by inheritance. They have the same attributes and methods derived from the entities (base class) such as HP, energy, id, owner etc. Each of the child class have it own unique attributes and methods.

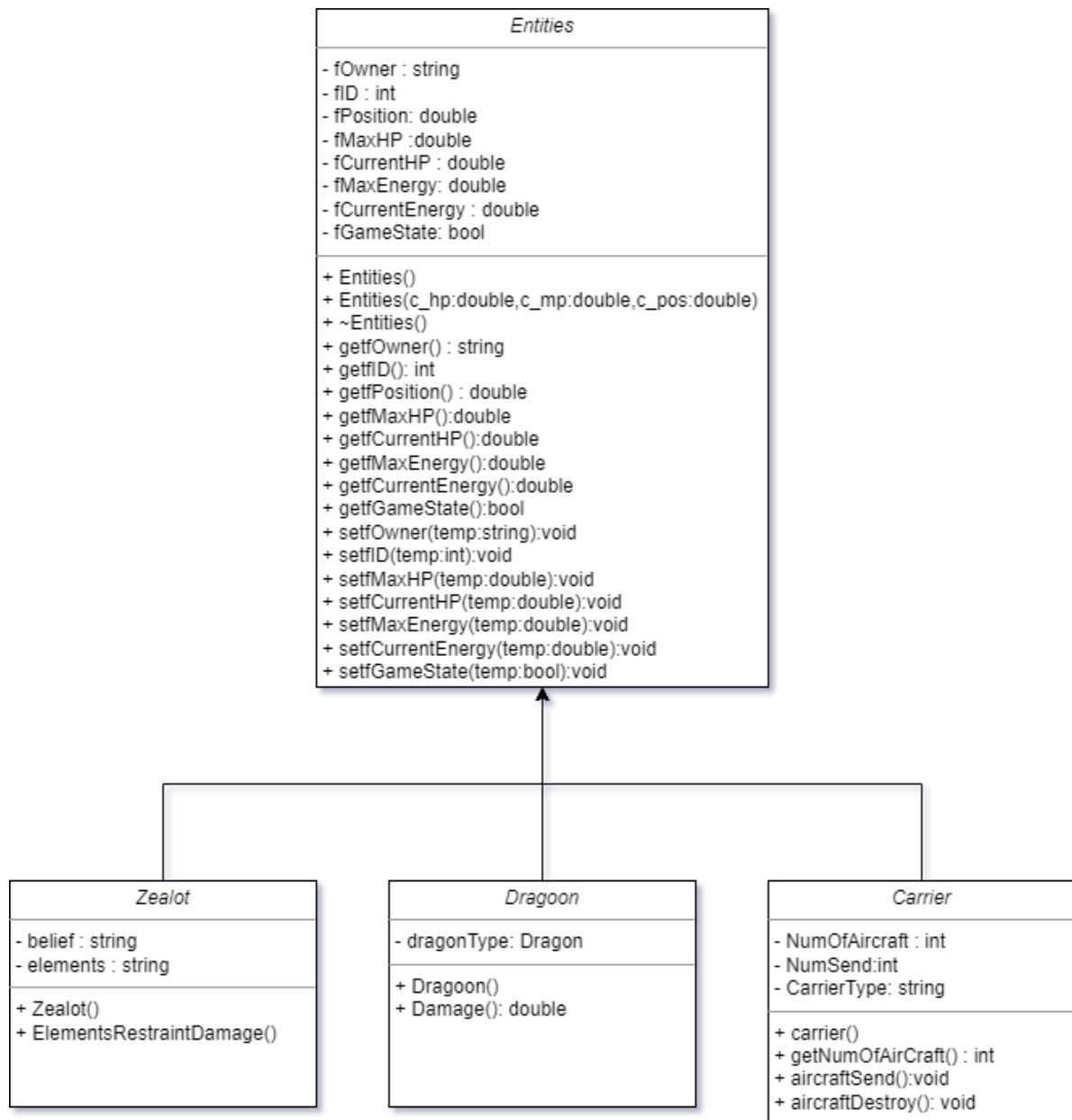
Zealots have its own attributes belief and elements, and methods to calculate elements counter damage such as water restraint fire so its damage will increase, else fire elements attack water elements, damage will decrease.

Dragoon has different attributes compare with another two-child class, dragon type and methods damage. Dragon type is used to store what dragon beast dragoon riding. And the damage calculation is different with the Zealots because dragoon doesn't have elements, so just simply do the add on calculation.

Carrier attributes is NumOfAircraft, NumSend, CarrierType. This is to store the number of aircraft on each carrier, number of aircraft that can send by carrier once time and the carrier type. The methods different with another derived class is used to perform send aircraft action and destroy aircraft.

Concept

Inheritance enables code reusability easily and saves time. It is declared by the characteristic of the class to do the classification. In inheritance, every child class have same attributes, methods, and fields with their base class, and they own unique attributes, fields, or methods.



Implementation

Entities.h

```
#pragma once
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
#include <sstream>
```

```
using namespace std;
```

```
class Entities {
```

```
private:
```

```
    string fOwner;
```

```
    int fID;
```

```
    double fPosition;
```

```
    double fMaxHP;
```

```
    double fCurrentHP;
```

```
    double fMaxEnergy;
```

```
    double fCurrentEnergy;
```

```
    bool fGameState;
```

```
    string fMessage;
```

```
public:
```

```
    Entities(); // constructor
```

```
    Entities(double c_hp, double c_mp, double c_pos); //overload constructor
```

```
with parameter
```

```
    ~Entities(); // destructore
```

```
    //getter
```

```
    string getfOwner();
```

```
    int getfID();
```

```
    double getfPosition();
```

```
    double getfMaxHP();
```

```
    double getfCurrentHP();
```

```
    double getfMaxEnergy();
```

```
    double getfCurrentEnergy();
```

```
    bool getfGameState();
```

```
    //setter
```

```
    void setfOwner(string temp);
```

```
    void setfID(int temp);
```

```
    void setfPostion(double temp);
```

```
    void setfMaxHP(double temp);
```

```
    void setfCurrentHP(double temp);
```

```
    void setfMaxEnergy(double temp);
```

```
    void setfCurrentEnergy(double temp);
```

```
    void setfGameState(bool temp);
```

```
};
```

Zealot.h

```
#pragma once
#include "Entities.h"
#include <iostream>

using namespace std;

class Zealot :public Entities{
private:
    string belief;
    string elements;
public:
    Zealot();
    void Print();
};
```

Carrier.h

```
#pragma once
#include "Entities.h"
#include <iostream>

using namespace std;

class Carrier :public Entities{
private:
    int NumOfAircraft;
    int NumSend;
    string CarrierType;
public:
    Carrier();
    void Print();
};
```

Dragoon.h

```
#pragma once
#include "Entities.h"
#include <iostream>

using namespace std;

class Dragoon :public Entities{
private:
    string DragonType;
public:
    Dragoon();
    void Print();
};
```


Entites.cpp

```
#include "Entities.h"
```

```
Entities::Entities() {  
    cout << "Hi, IM Concstructor" << endl;  
    fOwner = "Alex";  
    fID = 0;  
    fPosition = 0;  
    fMaxHP = 100;  
    fCurrentHP = 100;  
    fCurrentEnergy = 100;  
    fMaxEnergy = 100;  
    fGameState = false;  
}
```

```
Entities::Entities(double c_hp, double c_mp, double c_pos) {  
    fCurrentHP = c_hp;  
    fCurrentEnergy = c_mp;  
    fPosition = c_pos;  
    fOwner = "Alex";  
    fID = 12;  
    fMaxHP = 100;  
    fMaxEnergy = 100;  
    fGameState = false;  
}
```

```
Entities::~Entities() {  
    cout << "\nHi, IM destructor for " << endl;  
    this->PrintStatus();  
}
```

```
string Entities::getfOwner() {  
    return fOwner;  
}
```

```
int Entities::getfID() {  
    return fID;  
}
```

```
double Entities::getfCurrentHP() {  
    return fCurrentHP;  
}
```

```
double Entities::getfCurrentEnergy() {  
    return fCurrentEnergy;  
}
```

```
double Entities::getfMaxHP() {  
    return fMaxHP;  
}
```

```
double Entities::getfMaxEnergy() {  
    return fMaxEnergy;  
}
```

```
double Entities::getfPosition() {  
    return fPosition;  
}
```

```
bool Entities::getfGameState() {  
    return fGameState;  
}
```

```

void Entities::setfOwner(string temp) {
    fOwner = temp;
}

void Entities::setfCurrentHP(double temp) {
    fCurrentHP = temp;
}

void Entities::setfID(int temp) {
    fID = temp;
}

void Entities::setfMaxHP(double temp) {
    fMaxHP = temp;
}

void Entities::setfMaxEnergy(double temp) {
    fMaxEnergy = temp;
}

void Entities::setfCurrentEnergy(double temp) {
    fCurrentEnergy = temp;
}

void Entities::setfPostion(double temp) {
    fPosition = temp;
}

void Entities::setfGameState(bool temp) {
    fGameState = temp;
}

```

Zealot.cpp

```

#include "Zealot.h"

Zealot::Zealot() {
    belief = "Jesus";
    elements = "light";
}

void Zealot::Print() {
    cout << "Zealot of " << belief << " (" << elements << ")\n" << endl;
}

```

Carrier.cpp

```

#include "Carrier.h"

Carrier::Carrier() {
    NumOfAircraft = 100;
    NumSend = 20;
    CarrierType = "Type A";
}

void Carrier::Print() {
    cout << "Carrier " << CarrierType << " have " << NumOfAircraft << "
aircraft and send " << NumSend << " to fight \n";
}

```

Dragoon.cpp

```
#include "Dragoon.h"

Dragoon::Dragoon() {
    DragonType = "Red Dragon";
}

void Dragoon::Print() {
    cout << DragonType << " is ready to fight \n" << endl;
}
```

Main.cpp

```
#include "Entities.h"
#include "Zealot.h"
#include "Dragoon.h"
#include "Carrier.h"
int main() {

    Zealot z;
    Dragoon d;
    Carrier c;

    z.setfOwner("Alex");
    d.setfOwner("Benjamin");
    c.setfOwner("ZH");

    z.Print();
    d.Print();
    c.Print();

    return 0;
}
```

Output

```
Microsoft Visual Studio Debug Console
Zealot of Jesus (light)

Red Dragon is ready to fight

Carrier Type A have 100 aircraft and send 20 to fight

Hi, IM destructor for
---Current State---
Player ID : 0
Player Name : ZH
Position : 0
HP : 100
MP : 100

Hi, IM destructor for
---Current State---
Player ID : 0
Player Name : Benjamin
Position : 0
HP : 100
MP : 100

Hi, IM destructor for
---Current State---
Player ID : 0
Player Name : Alex
Position : 0
HP : 100
MP : 100

C:\Users\green\Desktop\Swinburne\Data Structure\ProblemSet1\Task2\Task_2\x64\Debug\Task_2.exe (process 30276) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Troubleshooting

No problem encountered in this task.

Task 3

Description

This task is to add new function PingStatus() to display the contents of its current fields, add new private string fMessage and public function called Listen() to store the fMessage, add new public function Tell() to display the content of the fMessage, create insertion friend overload version of the tell function and extraction friend overload version of the Listen function to announce it message and save the message.

Concept

Friend function allow to access all the non-public members of a class without using the getter, setter etc to perform action. With friend function, we can increase the versatility of overloading operators and it is reusable easily.

Implementation

Entities.h

```
void PintStatus(); // to print our player current state
void Listen(string temp); // use to store fmessage
void Tell(); //used to display message

//friends
friend istream& operator>>(istream& ist,Entities& object); // input data
friend ostream& operator<<(ostream& ost, const Entities& object); // output data
```

Entities.cpp

```
void Entities::PintStatus() {
    cout << "----Current State----" << endl;
    cout << "Player ID : " << fID << endl;
    cout << "Player Name : " << fOwner << endl;
    cout << "Position : " << fPosition << endl;
    cout << "HP : " << fCurrentHP << endl;
    cout << "MP : " << fCurrentEnergy << endl;
}

void Entities::Listen(string temp) {
    fMessage = temp;
}

void Entities::Tell() {
    cout << "Message : " << fMessage << endl;
}

istream& operator>>(istream& ist, Entities& object) {
    cout << "Enter fMessage (friend Operator Overload): ";
    getline(ist, object.fMessage);
    return ist;
}

ostream& operator<<(ostream& ost, const Entities& object) {
    ost << "Print fMessage (friend Operator Overload) : " << object.fMessage
    << endl;

    return ost;
}
```

Main.cpp

```
#include "Entities.h"
#include "Zealot.h"
#include "Dragoon.h"
#include "Carrier.h"
int main() {

    Entities e;
    string msg;

    e.PintStatus();

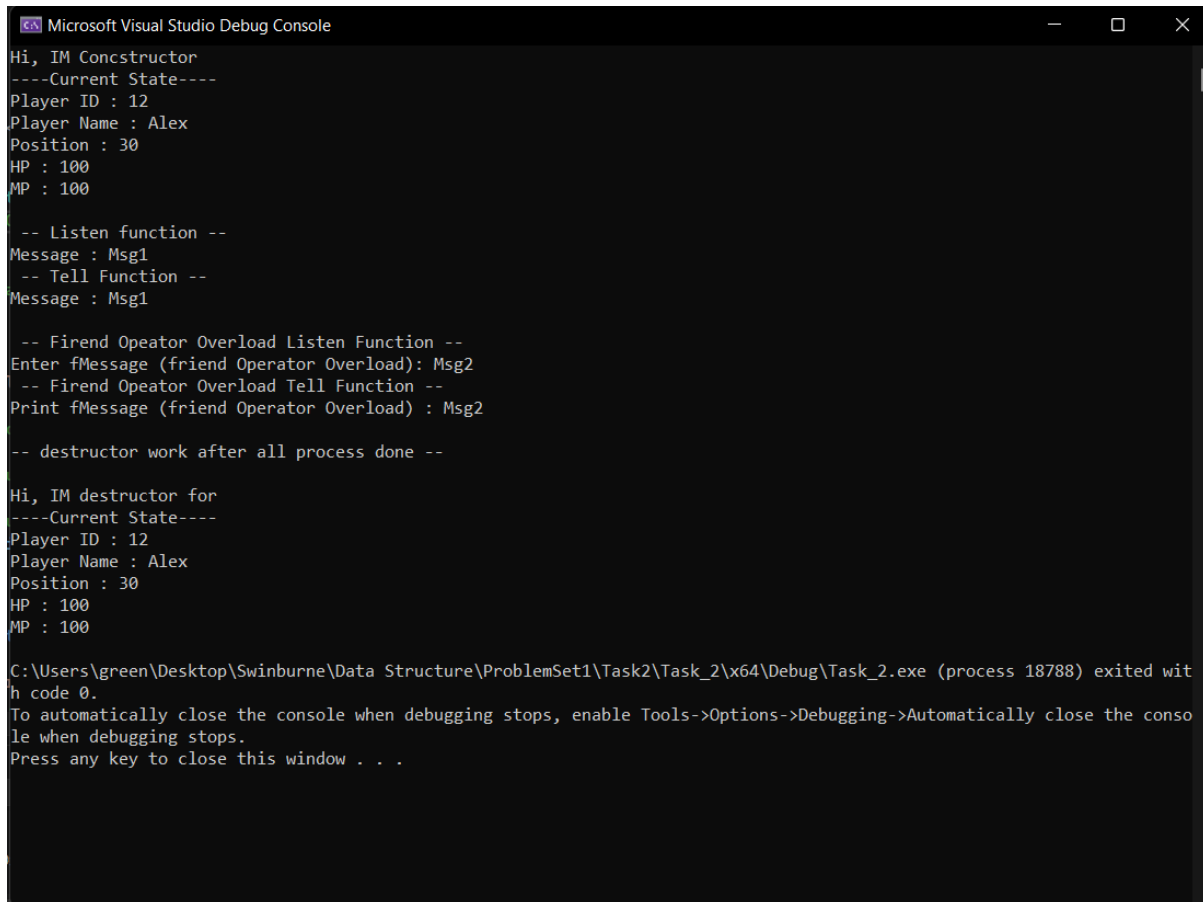
    cout << "\n -- Listen function -- " << endl;
    cout << "Message : "; getline(cin,msg);
    e.Listen(msg);
    cout << " -- Tell Function -- " << endl;
    e.Tell();

    cout << "\n -- Firend Opeator Overload Listen Function -- " << endl;
    cin >> e;
    cout << " -- Firend Opeator Overload Tell Function -- " << endl;
    cout << e;

    cout << " \n-- destructor work after all process done -- " << endl;

    return 0;
}
```

Output

A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the Visual Studio logo and the text "Microsoft Visual Studio Debug Console". The console output shows the execution of a C++ program. It starts with "Hi, IM Constructor", followed by "----Current State----" and a list of player attributes: "Player ID : 12", "Player Name : Alex", "Position : 30", "HP : 100", and "MP : 100". Then it shows function calls: "-- Listen function --", "Message : Msg1", "-- Tell Function --", "Message : Msg1", "-- Firend Opeator Overload Listen Function --", "Enter fMessage (friend Operator Overload): Msg2", "-- Firend Opeator Overload Tell Function --", and "Print fMessage (friend Operator Overload) : Msg2". This is followed by "-- destructor work after all process done --". Then it shows "Hi, IM destructor for", "----Current State----", and the same player attributes. At the bottom, it shows the program exit message: "C:\Users\green\Desktop\Swinburne\Data Structure\ProblemSet1\Task2\Task_2\x64\Debug\Task_2.exe (process 18788) exited with code 0." and instructions to enable automatic console closure in the Visual Studio options. The console text is white on a dark background.

```
Microsoft Visual Studio Debug Console

Hi, IM Constructor
----Current State----
Player ID : 12
Player Name : Alex
Position : 30
HP : 100
MP : 100

-- Listen function --
Message : Msg1
-- Tell Function --
Message : Msg1

-- Firend Opeator Overload Listen Function --
Enter fMessage (friend Operator Overload): Msg2
-- Firend Opeator Overload Tell Function --
Print fMessage (friend Operator Overload) : Msg2

-- destructor work after all process done --

Hi, IM destructor for
----Current State----
Player ID : 12
Player Name : Alex
Position : 30
HP : 100
MP : 100

C:\Users\green\Desktop\Swinburne\Data Structure\ProblemSet1\Task2\Task_2\x64\Debug\Task_2.exe (process 18788) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Troubleshooting

Research how to read a line of string in c++.

Task 4

Description

Use Extraction operator overload on fMessage , read the command input in the console and perform the appropriate action.

Concept

By using extraction operator overload to capture the message from the console, an units be able to communicate with the other unit to perform appropriate action.

Implementation

```
istream& operator>>(istream& ist, Entities& object) {
    //cout << "Enter fMessage (friend Operator Overload): ";
    //getline(ist, object.fMessage);

    vector<string> splitString; // create an vector to store array
    string temp; // create varaible to store input command temporary
    istringstream iss; // split string into word

    cout << "Heal HP by command (Example : HEAL 20) : ";
    getline(ist,temp); // read command
    iss.str(temp); // split string into word
    for (string s; iss >> s;) { // for loop until the end of the array in iss
        splitString.push_back(s); // push word splited by iss into
splitString(vector)
    }
    object.fMessage = temp; // store full command into fmessage
    object.fCurrentHP += stod(splitString[1]); // add up current hp and heal
hp
    return ist;
}

ostream& operator<<(ostream& ost, const Entities& object) {
    ost << "Print fMessage Command (friend Operator Overload) : " <<
object.fMessage << endl;
    ost << "Current HP : " << object.fCurrentHP << endl;
    return ost;
}
```


Output

```
----Current State----
Player ID : 12
Player Name : Alex
Position : 40
HP : 200
MP : 300

-- Listen function --
Message : msg hehe
-- Tell Function --
Message : msg hehe

-- Friend Operator Overload Listen Function --
Heal HP by command (Example : HEAL 20) : HEAL 100
-- Friend Operator Overload Tell Function --
Print fMessage Command (friend Operator Overload) : HEAL 100
Current HP : 300

-- destructor work after all process done --

Hi, IM destructor for
----Current State----
Player ID : 12
Player Name : Alex
Position : 40
HP : 300
MP : 300
```

Troubleshooting

Research for how to split string into array. Convert string into double.

Task 5

Description

Create viewpoint or window by using SFML.

Implementation

Character.h

```
#pragma once
#include <iostream>
#include "SFML/Graphics.hpp"

using namespace std;

class Character {
private:
    float left, right, up, down;
    sf::Keyboard keypress;
    sf::Texture image; // load the image use
    sf::IntRect imageRect; // get part of the image by using height width and
x y
    sf::Sprite player; // character
    sf::Clock clock; // clock
public:
    Character();
    sf::Sprite getCharacter();
    void CreateCharacter();
    void moveLeft();
    void moveRight();
    void moveUp();
    void moveDown();
    void move(int top);
    void Print();
};
```

Character.cpp

```
#include "Character.h";

Character::Character() {
    imageRect.left = 0; // sf::IntRect ss(left,top,width,height); start,
start,picture width,picture height
    imageRect.top = 0;
    imageRect.width = 400;
    imageRect.height = 600;
}

void Character::CreateCharacter() {
    image.loadFromFile("assets/picture/character_sprite.png"); // load image from
file
    player.setTexture(image); // set image
    player.scale(0.2f,0.2f); // set scale of image
    player.setPosition(400,200); // set starting position of image
    player.setTextureRect(imageRect); // get the part of image from image source
}

void Character::moveLeft() {
    move(1200);
    player.move(-0.2f,0);
}

void Character::moveRight() {
    move(1800);
```

```

        player.move(0.2f, 0);
    }
    void Character::moveUp() {
        move(600);
        player.move(0, -0.2f);
    }
    void Character::moveDown() {
        move(0);
        player.move(0, 0.2f);
    }

    void Character::move(int top) {
        imageRect.top = top;
        if (clock.getElapsedTime().asSeconds() > 0.5f) { // the refresh rate of the
character
            if (imageRect.left == 1200) // this if else is for detect the row of the
picture
                imageRect.left = 0;
            else
                imageRect.left += 400;

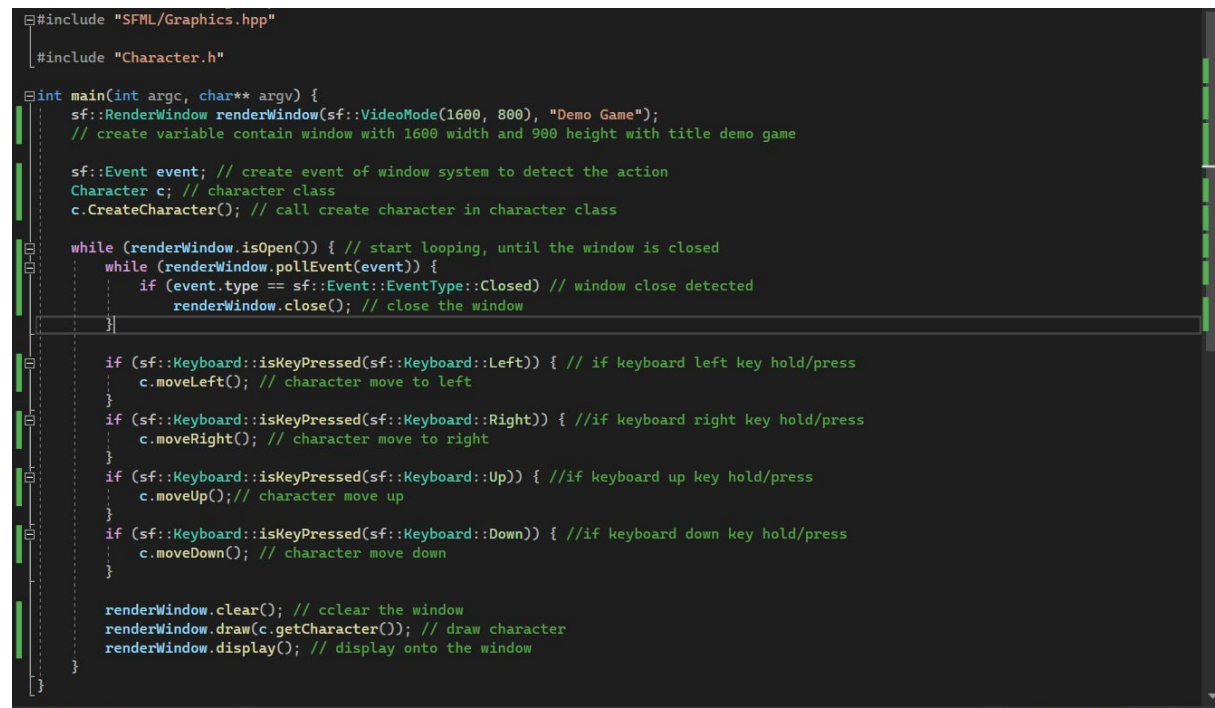
            player.setTextureRect(imageRect);
            clock.restart();
        }
    }

    sf::Sprite Character::getCharacter() { // return character
        return player;
    }

    void Character::Print() {
        cout << imageRect.left << imageRect.top << imageRect.width <<
imageRect.height << endl;
    }

```

Main.cpp



```

#include "SFML/Graphics.hpp"
#include "Character.h"

int main(int argc, char** argv) {
    sf::RenderWindow renderWindow(sf::VideoMode(1600, 800), "Demo Game");
    // create variable contain window with 1600 width and 800 height with title demo game

    sf::Event event; // create event of window system to detect the action
    Character c; // character class
    c.CreateCharacter(); // call create character in character class

    while (renderWindow.isOpen()) { // start looping, until the window is closed
        while (renderWindow.pollEvent(event)) {
            if (event.type == sf::Event::EventType::Closed) // window close detected
                renderWindow.close(); // close the window
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) { // if keyboard left key hold/press
            c.moveLeft(); // character move to left
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) { //if keyboard right key hold/press
            c.moveRight(); // character move to right
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) { //if keyboard up key hold/press
            c.moveUp(); // character move up
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) { //if keyboard down key hold/press
            c.moveDown(); // character move down
        }

        renderWindow.clear(); // clear the window
        renderWindow.draw(c.getCharacter()); // draw character
        renderWindow.display(); // display onto the window
    }
}

```

```

#include "SFML/Graphics.hpp"

#include "Character.h"

int main(int argc, char** argv) {
    sf::RenderWindow renderWindow(sf::VideoMode(1600, 800), "Demo Game");
    // create variable contain window with 1600 width and 900 height with title
    demo game

    sf::Event event; // create event of window system to detect the action
    Character c; // character class
    c.CreateCharacter(); // call create character in character class

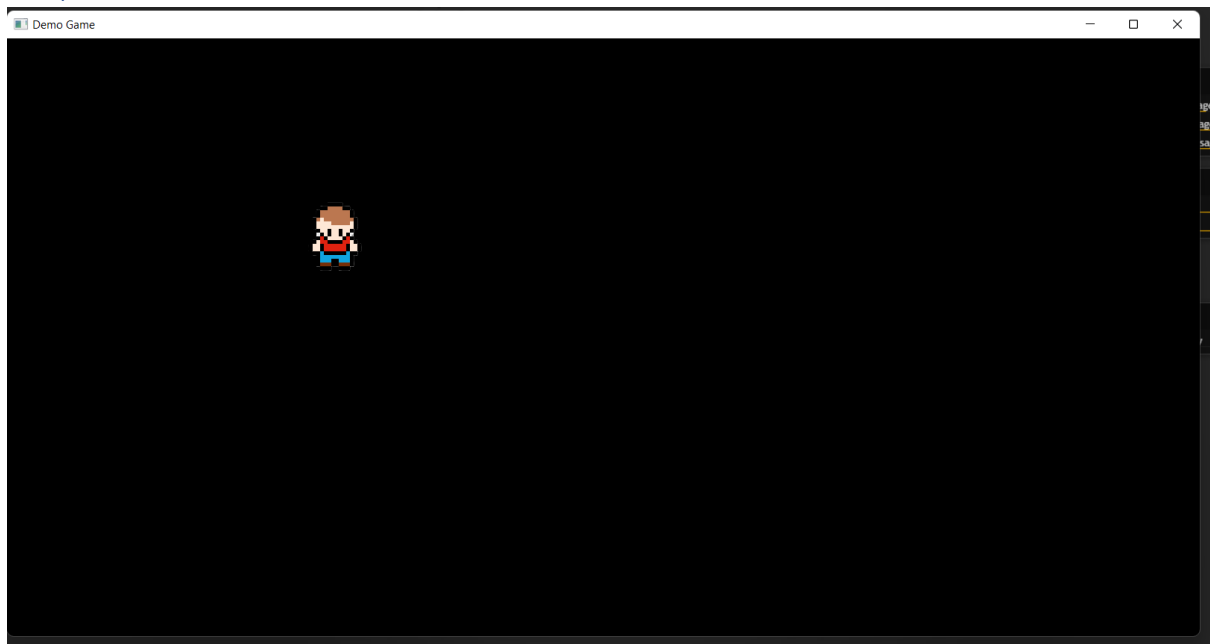
    while (renderWindow.isOpen()) { // start looping, until the window is closed
        while (renderWindow.pollEvent(event)) {
            if (event.type == sf::Event::EventType::Closed) // window close
detected
                renderWindow.close(); // close the window
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) { // if keyboard left
key hold/press
            c.moveLeft(); // character move to left
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) { //if keyboard
right key hold/press
            c.moveRight(); // character move to right
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) { //if keyboard up key
hold/press
            c.moveUp(); // character move up
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) { //if keyboard down
key hold/press
            c.moveDown(); // character move down
        }

        renderWindow.clear(); // cclear the window
        renderWindow.draw(c.getCharacter()); // draw character
        renderWindow.display(); // display onto the window
    }
}

```

Output



Troubleshooting

Research how to create render window to create a character that can change view in moving. Detect the arrow left, right ,top , down on keyboard.