COMP315

Project Documentation

INTERSTELLAR

Members	
1. Lethokuhle Buthelezi	(222021523)
2. Siphiwo Hlabisa	(222058571)
3. Lethukuthula Mthembu	(223041297)
4. Thubalihle Buthelezi	(220024270)
5. Thandokazi Phunzana	(222029499)
6. Samukelisiwe Myeza	(219074703)
7. Sthembile Kubheka	(221043970)
8. Nyandano Tshishonge	(223043642)
9. Lindokuhle Machangu	(221032601)
10.Jaan Fredericks	(221009036)

*** DELETE this, and the […] when using the template.

Introduction

CONCEPT OF THE GAME:

The game is an interactive adventure where players take on the role of a brave astronaut embarking on a journey to explore the galaxy. The player must use their hand-eye coordination, timing skills, spatial awareness, and knowledge to escape Earth's atmosphere by progressing through the different atmospheric layers. Each level presents unique challenges that require dodging misleading answers and avoiding obstacles within a time limit.

Program Development:

The game is designed to test and enhance players' knowledge about Earth's atmospheric layers through a series of questions presented in a marquee manner. Players use the up, down, left, and right buttons on the keyboard to move the astronaut character to select the correct answer by colliding with it. They must answer 8 questions correctly to advance to the next level. Each incorrect answer or collision with an obstacle results in the loss of a life, with players given 3 lives to sustain them throughout the game.

Level Introductions:

Level 1: Troposphere - Players must dodge misleading answers and walk towards the correct answers to questions about Earth's surface before time runs out.

Level 2: Stratosphere - Players navigate through misleading answers and answer questions about Earth's tropopause region while avoiding collisions with cloud obstacles.

Level 3: Mesosphere - Players face the deadliest atmospheric layer, dodging misleading answers to questions about Earth's mesosphere while surviving meteor showers.

Programming Techniques

1. Function

Screenshot:

```
//method to make sure question texts do not overlap out the window frame
string wrapText(const std::string& text, const sf::Font& font, unsigned int characterSize, float maxWidth) {
    std::stringstream wrapped;
    std::stringstream word;
    float lineWidth = 0;
    float spaceWidth = font.getGlyph(' ', characterSize, false).advance;

    for (char c : text) {
        if (c == ' ') {
            float wordWidth = font.getGlyph(word.str()[0], characterSize, false).advance * word.str().length();
            if (lineWidth + wordWidth > maxWidth) {
                wrapped << "\n"; // Move to next line
                lineWidth = 0;
            }
            wrapped << word.str() << " ";
            lineWidth += wordWidth + spaceWidth;
            word.str(""); // Clear buffer
        }
        else {
            word << c;
        }
    }
    wrapped << word.str(); // Append last word
    return wrapped.str();
}
```

Motivation:

We implemented the deductLives function to streamline life deduction logic, enabling code reuse throughout the main class whenever a player collides with an obstacle. By encapsulating this logic, we gained controlled access to private variables while making the code more readable, maintainable, and efficient. This approach avoided hardcoding and repetition, allowing for easier modifications and updates in the future.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	The function runs efficiently and is reusable also easy to edit.

2. Class

Screenshot:

```cpp
#include "Timer.h"
#include <iomanip>  // For formatting
#include <sstream>
#include <iostream>
#include <string>
using namespace std;
sf::Clock myclock;

Timer::Timer() {
    timer = sf::seconds(120);
    myclock.restart();
}


string Timer::startTimer(bool& quizEnded) {

    timeElapsed = myclock.getElapsedTime();
        timeRemaining = timer - timeElapsed;

        if (timeRemaining <= sf::seconds(0)) { // Timer has run out
            quizEnded = true; //quiz ends when time runs up
            return "00:00";
        }
        if (quizEnded == true && timeRemaining > sf::seconds(0)) {
            totalTime += timeRemaining;

        }
        return formatTime(timeRemaining);
}

void Timer::resetTimer() {
    myclock.restart();
}
```

Motivation:

We implemented the timer functionality as a class to bring structure and organization to the code, making it more maintainable and efficient. By encapsulating the timer's data and behavior within a single unit, we ensured that the code is easy to understand. This approach also allowed us to hide internal implementation details, reducing the risk of errors and improving overall code quality. With the timer class, we can confidently manage time-related functionality, format time displays, and even perform operations like adding timers together, all while keeping the code clean and scalable.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	

3. Struct

Screenshot:

```
public:
    struct Question {
        string questionText;
        vector<string> answers;
        int correctAnswerIndex;
    };
```

Motivation:

By defining a struct for our questions, we were able to neatly bundle together the question text, possible answers, and the correct answer index. This made our code more organized and easier to work with, allowing us to focus on building a great quiz experience. Plus, the struct's constructor helped ensure that our question data was always valid and consistent

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	By using a struct, we've achieved organized, readable, and maintainable code that supports the quiz game's functionality.

4. Pointer

Screenshot:

```
result.timeElapsed = this->timeElapsed + other.timeElapsed;
result.timer = this->timer + other.timer;
```

Motivation:

We used a special pointer called “this”, that refers to the current timeLapsed object and the current timer object. Copying was avoided when we needed to modify the result object by accessing the objects properties directly

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially	x	
Completely		

5. Reference

Screenshot:

```
public:
    StartGameButton(const std::string& imagePath, const sf::Vector2f& position,float imageResizer) {

        if (!StartButtonTexture.loadFromFile(imagePath)) {
            std::cerr << "Error loading imgae\n";
        }
        StartButtonSprite.setTexture(StartButtonTexture);
        StartButtonSprite.setPosition(position);
        StartButtonSprite.setScale(imageResizer,imageResizer);

    }
```

StartGameButton.cpp from line 16

Motivation:

We used const auto& in our loop to avoid making unnecessary copies of the data, which keeps things running smoothly. The const part also gives us peace of mind, ensuring we don't accidentally mess with the data we're working with. It helps keep our code efficient and reliable

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	we recognize that the repeated code blocks can be further optimized by encapsulating the logic within a function, which would improve code maintainability and reusability

6. Vector

Screenshot:

```
if (showHomeWindow == 4) {
    questionText.setString(wrapText(quiz.getQuestion1(),font,22,400));
    vector<string> answers1(4);
    answers1 = quiz.getAnswers1();
    for (int i = 0; i < answers1.size(); i++) {
        answerText[i].setString(answers1[i]);

    }
}
```

Motivation:

We utilized a vector to manage questions due to its dynamic nature, allowing for efficient storage and retrieval of quiz content. This data structure enabled us to easily add, remove, or modify questions, making it ideal for our quiz game. By using a vector, we ensured our game could handle a large number of questions while maintaining performance and scalability, ultimately delivering a seamless and engaging experience for users.it also made it easy to implement question and randomize answers

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely,

Not met		**provide a short explanation to support the claim**
Partially		
Completely	X	Yes because we were able to manage the questions effeciently

7. Data Structures

Screenshot:

```
//For playing background music
void playMusicForWindow(int Window) {
    map<int, string> musicMap = {
   {1, "FurElisePiano.mp3"},
   {2,"RocketLaunchSoundEffect_0.mp3."},
   {3, "FurElisePiano.mp3"},
   {4, "Earthlevel.mp3"},
   {5,"RocketLaunchSoundEffect.mp3"},
   {6, "FurElisePiano.mp3"},
   {8,"RocketLaunchSoundEffect_0.mp3"},
   {9, "FurElisePiano.mp3"},
   {10, "rocklevel.wav"},
   {11,"PABoutroSong.mp3"},
   {12,"retryLevelSound.mp3"}
    };
```

Motivation:

We used a map to organize our music files because it's super helpful for linking specific songs to certain levels or events in the game. With a map, we can easily find the right music file when we need it, which makes managing the soundtrack a breeze. Plus, it's really flexible, so if we want to add or remove music tracks later on, it's no big deal.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	We successfully utilized the map data structure to efficiently store and retrieve music files based on specific level identifiers, demonstrating its effectiveness in organizing and managing key-value pairs in our game.

8. Class Template

Screenshot:

[*Provide a screenshot of your code. This will be code of a class template in your game*]

Motivation:
[*Explain why you decided to specifically use a class template in this area of your code, and the functionality it provides to your code.*]

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met	X	
Partially		
Completely		

9. Function Template

Screenshot:

[*Provide a screenshot of your code. This will be code of a function template in your game*]

Motivation:

[*Explain why you decided to specifically use a function template in this area of your code, and the functionality it provides to your code.*]

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met	X	
Partially		
Completely		

10. Operator Overloading

Screenshot:

```
Timer Timer::operator+(const Timer& other) const {
    Timer result;
    result.timeElapsed = this->timeElapsed + other.timeElapsed;
    result.timer = this->timer + other.timer;
    return result;
}
```

Timer class

Motivation:

We utilized operator overloading for the Timer class to enable intuitive addition of Timer objects. By overloading the '+' operator, we can easily combine two Timer instances, resulting in a new Timer object that represents the cumulative time. This approach enhances code readability and usability, allowing us to write more expressive and natural-looking code when working with time-related calculations.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	Our implementation correctly calculates the cumulative time, ensuring accurate results.

11. Object Oriented Programming

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide evidence to support the claim
Not met		
Partially		
Completely	X	Our code has different class making it easy to edit code and fix problems without affecting the entire system