```cpp

#ifdef _DEBUG
#pragma comment(lib,"sfml-graphics-d.lib")
#pragma comment(lib,"sfml-audio-d.lib")
#pragma comment(lib,"sfml-system-d.lib")
#pragma comment(lib,"sfml-window-d.lib")
#pragma comment(lib,"sfml-main-d.lib")
#pragma comment(lib,"sfml-network-d.lib")
#else
#pragma comment(lib,"sfml-graphics.lib")
#pragma comment(lib,"sfml-audio.lib")
#pragma comment(lib,"sfml-system.lib")
#pragma comment(lib,"sfml-main.lib")
#pragma comment(lib,"sfml-window.lib")
#pragma comment(lib,"sfml-network.lib")
#endif
#pragma comment(lib,"opengl32.lib")
#pragma comment(lib,"glu32.lib")

#include <SFML/Graphics.hpp>
#include "SFML/OpenGL.hpp"
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
#include "Game.h"
#include "Licence.h"
#include "SplashScreen.h"
/*
Written by Peter Lowe May 2015
Total Project Time ~ 10 hours*/


int main()
{
    Game game;
    game.run();
}

float Game::s_screenWidth = 600;
float Game::s_screenHeight = 400;
GameState Game::currentState = GameState::Licence;

/// <summary>
/// @brief main game constructor creating the render window with settings
/// </summary>
Game::Game() : m_window(sf::VideoMode(static_cast<unsigned>(Game::s_screenWidth), static_cast<unsigned>(Game::s_screenHeight)), "MarioKart",sf::Style::Default)
{
    loadContent();
    m_window.setKeyRepeatEnabled(false);
}
/// <summary>
```

```cpp
/// @brief load the font and initialise everything else.
///
/// after loading the font pass a refrence to font resourse to other class iniatilisers
/// </summary>
void Game::loadContent()
{
    int m_gold;
    if (!m_arialFont.loadFromFile("ASSETS/FONTS/BebasNeue.otf"))
    {
        std::cout << "error with font file file";
    }



    m_licenceScreen.initialise(m_arialFont);
    m_splashScreen.initialise(m_arialFont);

    m_mainMenu.initialise(m_arialFont);
    m_mainGame.initialise();
    m_helpPage.initialise(m_arialFont);

    m_gold = 0;
#ifdef STARTRICH
    m_gold = 1000;
#endif // STARTRICH


#ifdef TEST_FPS
    x_updateFrameCount = 0;
    x_drawFrameCount = 0;
    x_secondTime = sf::Time::Zero;
    x_updateFps.setFont(m_arialFont);
    x_updateFps.setPosition(20, 300);
    x_updateFps.setCharacterSize(24);
    x_updateFps.setFillColor(sf::Color::White);
    x_drawFps.setFont(m_arialFont);
    x_drawFps.setPosition(20, 350);
    x_drawFps.setCharacterSize(24);
    x_drawFps.setFillColor(sf::Color::White);
#endif // TEST_FPS
}

/// <summary>
/// @brief main game loop.
/// </summary>
void Game::run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;
    sf::Time timePerFrame = sf::seconds(1.f / 60.f);
    while (m_window.isOpen())
```

```
103     {
104         processEvents();
105         timeSinceLastUpdate += clock.restart();
106         while (timeSinceLastUpdate > timePerFrame)
107         {
108             timeSinceLastUpdate -= timePerFrame;
109 
110             processEvents();
111             update(timePerFrame);
112 #ifdef TEST_FPS
113             x_secondTime += timePerFrame;
114             x_updateFrameCount++;
115             if (x_secondTime.asSeconds() > 1)
116             {
117                 char bufferDps[256];
118                 char bufferUps[256];
119                 sprintf_s(bufferUps, "%d UPS", x_updateFrameCount-1);
120                 x_updateFps.setString(bufferUps);
121                 sprintf_s(bufferDps, "%d DPS", x_drawFrameCount);
122                 x_drawFps.setString(bufferDps);
123                 x_updateFrameCount = 0;
124                 x_drawFrameCount = 0;
125                 x_secondTime = sf::Time::Zero;
126             }
127 #endif // TEST_FPS
128         }
129         render();
130 #ifdef TEST_FPS
131         x_drawFrameCount++;
132 #endif // TEST_FPS
133     }
134 }
135 /// <summary>
136 /// @brief call the appropiate processEvents method ofr currentstate
137 /// </summary>
138 void Game::processEvents()
139 {
140     sf::Event event;
141     while (m_window.pollEvent(event))
142     {
143         if (event.type == sf::Event::Closed)
144         {
145             m_window.close();
146         }
147         switch (currentState)
148         {
149         case GameState::Licence:
150             break;
151         case GameState::Splash:
152             m_splashScreen.processInput(event);
153             break;
154         case GameState::MainMenu:
```

```
155             break;
156         case GameState::Help:
157             m_helpPage.processInput(event);
158             break;
159         case GameState::Game:
160             m_mainGame.processInput(event);
161             break;
162         default:
163             break;
164         }
165     }
166 }
167 /// <summary>
168 /// @brief call the update method corresponding to the current game state
169 /// </summary>
170 /// <param name="time">update delta time</param>
171 void Game::update(sf::Time time)
172 {
173     switch (currentState)
174     {
175     case GameState::Licence:
176         m_licenceScreen.update(time);
177         break;
178     case GameState::Splash:
179         m_splashScreen.update(time);
180         break;
181     case GameState::MainMenu:
182         m_mainMenu.update(time, m_window);
183         break;
184     case GameState::Help:
185         m_helpPage.update(time);
186         break;
187     case GameState::Game:
188         m_mainGame.update(time);
189         break;
190     default:
191         break;
192     }
193 }
194
195 /// <summary>
196 /// @brief call the renderer for the current game state
197 /// </summary>
198 void Game::render()
199 {
200     m_window.clear();
201     switch (currentState)
202     {
203     case GameState::Licence:
204         m_licenceScreen.render(m_window);
205         break;
206     case GameState::Splash:
```

```
207             m_splashScreen.render(m_window);
208             break;
209         case GameState::MainMenu:
210             m_mainMenu.render(m_window);
211             break;
212         case GameState::Help:
213             m_helpPage.render(m_window);
214             break;
215         case GameState::Game:
216             m_mainGame.render(m_window);
217             break;
218         default:
219             break;
220         }
221     #ifdef TEST_FPS
222         m_window.draw(x_updateFps);
223         m_window.draw(x_drawFps);
224     #endif // TEST_FPS
225         m_window.display();
226     }
```