```cpp
#include "Game.h"
#include <iostream>
#include <time.h>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Main.hpp>
#include <SFML/System/Time.hpp>
/// <summary>
/// @author Peter Lowe
/// @date May 2016
/// @version 1.0
///
/// time 12 hours
/// </summary>




/// <summary>
/// @brief default constructor.
///
/// create a new window and initialise member objects
/// </summary>
Game::Game()
	: m_window(sf::VideoMode(800, 480), "Simon game by Pete")
	, m_redSquare(sf::Vector2f(200, 200))
	, m_yellowSquare(sf::Vector2f(200, 200))
	, m_greenSquare(sf::Vector2f(200, 200))
	, m_blueSquare(sf::Vector2f(200, 200))
	, m_blueTimer(0)
	, m_redTimer(0)
	, m_yellowTimer(0)
	, m_greenTimer(0)
	, m_flashTime(15)
	,m_currentGameMdoe(GameMode::Starting)
	, m_noteSequence()
	,m_inputTime()
{

#ifndef NO_RANDOM
	// always play the same game when debugging
	std::srand(time(NULL));
#endif // NO_RANDOM
	//setup the default values for everything else
	setupButtons();
	resetButtons();
}

/// <summary>
/// @brief reset buttons to false.
///
/// run this before processing events again after an update to deal with the button
presses
/// </summary>
void Game::resetButtons()
{
	m_blueButtonPressed = false;
	m_redButtonPressed = false;
	m_yellowButtonPressed = false;
	m_greenButtonPressed = false;
}
```

```
/// <summary>
/// @brief get a new sequence of notes.
/// </summary>
void Game::randomiseNotes()
{
	for (int i = 0; i < 32; i++)
	{
		// looking for values of 0,1,2,3
		m_noteSequence[i] = std::rand() % 4;
	}
	m_inputTime = sf::seconds( 0 );
}

/// <summary>
/// @brief setup text, squares and sounds
///
/// load the font and sound
/// setup the text messages
/// setup the tone (pitch)
/// position and colour of the squares.
/// </summary>
void Game::setupButtons()
{
	// set colour and location of each square
	m_greenSquare.setFillColor(GREEN);
	m_greenSquare.setPosition(sf::Vector2f(350, 30));
	m_redSquare.setFillColor(RED);
	m_redSquare.setPosition(sf::Vector2f(570, 30));
	m_yellowSquare.setFillColor(YELLOW);
	m_yellowSquare.setPosition(sf::Vector2f(350, 250));
	m_blueSquare.setFillColor(BLUE);
	m_blueSquare.setPosition(sf::Vector2f(570, 250));
	// load the sound file in a buffer
	if (m_toneBuffer.loadFromFile("assets/audio/tone.wav"))
	{
		std::cout << "beep loaded ok" << std::endl;
	}
	// assign the buffer to sounds and change pitch
	m_blueTone.setBuffer(m_toneBuffer);
	m_redTone.setBuffer(m_toneBuffer);
	m_redTone.setPitch(0.85f);
	m_yellowTone.setBuffer(m_toneBuffer);
	m_yellowTone.setPitch(0.7f);
	m_greenTone.setBuffer(m_toneBuffer);
	m_greenTone.setPitch(0.55f);
	// load the font file
	if (m_impactFont.loadFromFile("assets/fonts/impact.ttf"))
	{
		std::cout << "font loaded ok" << std::endl;
	}
	//setup the title
	m_titleText.setFont(m_impactFont);
	m_titleText.setColor(WHITE);
	m_titleText.setCharacterSize(64);
	m_titleText.setPosition(50, 30);
	m_titleText.setString("S I M O N");

	// setup green message easy game 8
	m_instructionsTextGreen.setFont(m_impactFont);
	m_instructionsTextGreen.setColor(GREEN);
	m_instructionsTextGreen.setCharacterSize(32);
	m_instructionsTextGreen.setPosition(50, 100);
	m_instructionsTextGreen.setString("Press green for \n an easy game");
	// set red text for medium game 16
	m_instructionsTextRed.setFont(m_impactFont);
	m_instructionsTextRed.setColor(RED);
	m_instructionsTextRed.setCharacterSize(32);
	m_instructionsTextRed.setPosition(50, 200);
	m_instructionsTextRed.setString("Press red for \n a medium game");
	// setup yellow text  for hard game 32
	m_instructionsTextYellow.setFont(m_impactFont);
	m_instructionsTextYellow.setColor(YELLOW);
	m_instructionsTextYellow.setCharacterSize(32);
	m_instructionsTextYellow.setPosition(50, 300);
	m_instructionsTextYellow.setString("Press yellow for \n a hard game");
	// setup status font
	m_instructionsTextBlue.setFont(m_impactFont);
	m_instructionsTextBlue.setColor(BLUE);
	m_instructionsTextBlue.setCharacterSize(32);
	m_instructionsTextBlue.setPosition(50, 400);
	m_instructionsTextBlue.setString("Press blue to \nexit game");

	m_statusText.setFont(m_impactFont);
	m_statusText.setColor(WHITE);
	m_statusText.setCharacterSize(22);
	m_statusText.setPosition(500, 453);
	m_statusText.setString(""); // no status on menu screen

}

/// <summary>
/// Main game entry point runs till game is finsihed
/// </summary>
void Game::run()
{
	sf::Clock clock;
	sf::Time timeSinceLastUpdate = sf::seconds(0);
	sf::Time timePerFrame = sf::seconds(1.f / 60.f);
	while (m_window.isOpen())
	{
		processEvents();
		// we need to process events so window can behave normally but we will
		// have multuiple process events per update. In this game we are 
assuming
		// each button can only be pressed once per update but multiple buttons
		// can be simulteanously be pressed, in which case the order is 
predetermined
		// by the order of the if statements
		timeSinceLastUpdate += clock.restart();
		while (timeSinceLastUpdate > timePerFrame)
		{
			timeSinceLastUpdate -= timePerFrame;
			processEvents();
			update(timePerFrame);
		}
		render();
	}
}
/// <summary>
/// @brief check for events
/// 
/// allows window to function and exit
/// then pass events on to own own games preocess events method
/// </summary>
```

```
void Game::processEvents()
{
	sf::Event event;
	while (m_window.pollEvent(event))
	{
		if (event.type == sf::Event::Closed)
		{
			m_window.close();
		}
		processGameEvents(event);
	}
}

/// <summary>
/// @brief detect buttons clicks.
///
/// detect the mouse button release event (for either button)
/// then check x co-ordinate for column and y corrdinate for row
/// if it's inside a button set the corresponding boolean
/// </summary>
/// <param name="event">system event</param>
void Game::processGameEvents(sf::Event& event)
{
	const int COL_1_LEFT = 350;
	const int COL_1_RIGHT = 550;
	const int COL_2_LEFT = 570;
	const int COL_2_RIGHT = 770;
	const int ROW_1_TOP = 20;
	const int ROW_1_BOTTOM = 230;
	const int ROW_2_TOP = 250;
	const int ROW_2_BOTTOM = 450;

	// check if the event is a a mouse button release
	if (sf::Event::MouseButtonReleased == event.type)
	{
		//check if its on the first col
		if (event.mouseButton.x > COL_1_LEFT && event.mouseButton.x <
COL_1_RIGHT)
		{
			//check which row
			if (event.mouseButton.y > ROW_1_TOP && event.mouseButton.y <
ROW_1_BOTTOM)
			{
				m_greenButtonPressed = true;
			}
			if (event.mouseButton.y > ROW_2_TOP && event.mouseButton.y <
ROW_2_BOTTOM)
			{
				m_yellowButtonPressed = true;
			}
		}
		// check if its on the scecond col
		if (event.mouseButton.x > COL_2_LEFT && event.mouseButton.x <
COL_2_RIGHT)
		{
			//check which row
			if (event.mouseButton.y > ROW_1_TOP && event.mouseButton.y <
ROW_1_BOTTOM)
			{
				m_redButtonPressed = true;
			}
```

```
				if (event.mouseButton.y > ROW_2_TOP && event.mouseButton.y < 
ROW_2_BOTTOM)
				{
					m_blueButtonPressed = true;
				}
			}
		}
}
/// <summary>
/// switch between the dedicated update methods for the game modes
/// </summary>
/// <param name="time">update delta time</param>
void Game::update(sf::Time time)
{
	switch (m_currentGameMdoe)
	{
	case GameMode::Starting:
		startingUpdate();
		break;
	case GameMode::Showing:
		showingUpdate();
		break;
	case GameMode::Recieving:
		recievingUpdate(time);
		break;
	case GameMode::GameOver:
		overUpdate();
		break;
	default:
		break;
	}
	// reset the booleans after update before next process events call
	resetButtons();
}
/// <summary>
/// @brief update game from menu.
/// 
/// using the four colour buttons the user can select
/// an easy,medium or hard game 8,16,32 notes
/// or blue to exit the game
/// 
/// </summary>
void Game::startingUpdate()
{
	m_statusText.setString("");
	if (m_blueButtonPressed)
	{
		m_window.close();
	}
	if (m_greenButtonPressed)
	{
		randomiseNotes();
		m_currentGameMdoe = GameMode::Showing;
		m_currentCount = 1;
		m_currentNote = 0;
		m_difficultyLevel = 8;
		m_flashTime = 30;
		m_modeChangeTimer = 0;
	}
	if (m_redButtonPressed)
	{
		randomiseNotes();
```

```
			m_currentGameMdoe = GameMode::Showing;
			m_currentCount = 1;
			m_currentNote = 0;
			m_difficultyLevel = 16;
			m_flashTime = 30;
			m_modeChangeTimer = 0;
		}
		if (m_yellowButtonPressed)
		{
			randomiseNotes();
			m_currentGameMdoe = GameMode::Showing;
			m_currentCount = 1;
			m_currentNote = 0;
			m_difficultyLevel = 32;
			m_flashTime = 30;
			m_modeChangeTimer = 0;
		}
}
/// <summary>
/// decrement each colours timer and then reset the colour on the button
/// </summary>
void Game::countdownTimers()
{
		if (m_blueTimer > 0)
		{
			if (0 == --m_blueTimer)
			{
				m_blueSquare.setFillColor(BLUE);
			}
		}
		if (m_redTimer > 0)
		{
			if (0 == --m_redTimer)
			{
				m_redSquare.setFillColor(RED);
			}
		}
		if (m_yellowTimer > 0)
		{
			if (0 == --m_yellowTimer)
			{
				m_yellowSquare.setFillColor(YELLOW);
			}
		}
		if (m_greenTimer > 0)
		{
			if (0 == --m_greenTimer)
			{
				m_greenSquare.setFillColor(GREEN);
			}
		}
}

/// <summary>
/// @brief check button presses against current note and act.
///
/// use 2 bools to check for a correct or incorrect click (there nay be no click{most
of the time})
/// if correct move onto next note and check if finished
/// if mistake then gameover and set win to false
/// </summary>
/// <param name="time">delta update time</param>
```

```cpp
void Game::recievingUpdate(sf::Time time)
{
	bool correct = false;
	bool mistake = false;

	m_statusText.setString("Listening");
	if (m_greenButtonPressed)
	{
		m_greenSquare.setFillColor(m_greenSquare.getFillColor() + sf::Color(64, 
64, 64, 255));
		m_greenTimer = m_flashTime;
		m_greenTone.play();
		if (0 == m_noteSequence[m_currentNote])
		{
			correct = true;
		}
		else
		{
			mistake = true;
		}
	}
	if (m_redButtonPressed)
	{
		m_redSquare.setFillColor(m_redSquare.getFillColor() + sf::Color(64, 64, 
64, 255));
		m_redTimer = m_flashTime;
		m_redTone.play();
		if (1 == m_noteSequence[m_currentNote])
		{
			correct = true;
		}
		else
		{
			mistake = true;
		}
	}
	if (m_yellowButtonPressed)
	{
		m_yellowSquare.setFillColor(m_yellowSquare.getFillColor() + 
sf::Color(64, 64, 64, 255));
		m_yellowTimer = m_flashTime;
		m_yellowTone.play();
		if (2 == m_noteSequence[m_currentNote])
		{
			correct = true;
		}
		else
		{
			mistake = true;
		}
	}
	if (m_blueButtonPressed)
	{
		m_blueSquare.setFillColor(m_blueSquare.getFillColor() + sf::Color(64, 
64, 64, 255));
		m_blueTimer = m_flashTime;
		m_blueTone.play();
		if (3 == m_noteSequence[m_currentNote])
		{
			correct = true;
		}
		else
```

```
            {
                mistake = true;
            }
        }
        if (correct)
        {
            m_currentNote++;
            if (m_currentNote == m_currentCount)
            {
                if (m_currentCount == m_difficultyLevel)
                {
                    m_currentGameMdoe = GameMode::GameOver;
                    m_win = true;
                    m_modeChangeTimer = 210;
                }
                else
                {
                    m_currentCount++;
                    m_currentNote = 0;
                    m_currentGameMdoe = GameMode::Showing;
                    m_modeChangeTimer = 60;
                    m_statusText.setString("...");
                    m_flashTime--;
                    if (m_flashTime < 10)
                    {
                        m_flashTime = 10;
                    }
                }
            }

        }
        if (mistake)
        {
            m_currentGameMdoe = GameMode::GameOver;
            m_win = false;
            m_modeChangeTimer = 120;
        }
        if (!correct && !mistake)
        {
            m_inputTime += time;
            if (m_inputTime.asMilliseconds() > 1500)
            {
                // extra delay for the first note
                if (m_inputTime.asMilliseconds() > 3000  || m_currentNote != 0)
                {
                    m_currentGameMdoe = GameMode::GameOver;
                    m_win = false;
                    m_modeChangeTimer = 120;
                }
            }
        }
        else
        {
            m_inputTime = sf::seconds(0);
        }
        countdownTimers();

    }

    /// <summary>
    /// @brief update the display of notes.
    ///
```

```
/// wait for a delay initiall, give the human a break between modes
/// then set the status text and if the previous note is finished (light gone out)
/// may still be playing then switch on the next note
/// green is 0
/// red is 1
/// yellow is 2
/// blue is 3
/// play the tone and highlight the button and set the timer
/// /// </summary>
void Game::showingUpdate()
{
	if (m_modeChangeTimer > 0)
	{
		m_modeChangeTimer--;
	}
	else
	{
		m_statusText.setString("Playing");
		if (0 == m_blueTimer && 0 == m_greenTimer && 0 == m_redTimer && 0 ==
m_yellowTimer)
		{
			if (m_currentNote < m_currentCount)
			{
				switch (m_noteSequence[m_currentNote])
				{
				case 0:
					m_greenTone.play();
					m_greenTimer = m_flashTime;

	m_greenSquare.setFillColor(m_greenSquare.getFillColor() + sf::Color(64, 64, 64,
255));
					break;
				case 1:
					m_redTone.play();
					m_redTimer = m_flashTime;
					m_redSquare.setFillColor(m_redSquare.getFillColor()
+ sf::Color(64, 64, 64, 255));
					break;
				case 2:
					m_yellowTone.play();
					m_yellowTimer = m_flashTime;

	m_yellowSquare.setFillColor(m_yellowSquare.getFillColor() + sf::Color(64, 64,
64, 255));
					break;
				case 3:
					m_blueTone.play();
					m_blueTimer = m_flashTime;

	m_blueSquare.setFillColor(m_blueSquare.getFillColor() + sf::Color(64, 64, 64,
255));
					break;
				default:
					break;
				}
				m_currentNote++;
			}
			else
			{
				// when all the notes have been played switch to listening
mode
				// and start back at the start of the sequence
```

```
					m_currentGameMdoe = GameMode::Recieving;
					m_currentNote = 0;
				}
			}
		}
		countdownTimers();
}

/// <summary>
/// @brief update game over.
/// 
/// run down countdowntimer and play tone for victory or defeat
/// when finsihed switch mode to starting
/// </summary>
void Game::overUpdate()
{
		// play same tone for defeat and set status message
		if (!m_win)
		{
			m_statusText.setString("Game Over you Lost");
			if (m_modeChangeTimer-- > 0)
			{
				if (m_modeChangeTimer%25 == 0)
				{
					m_greenTone.play();
				}
			}
			else
			{
				m_currentGameMdoe = GameMode::Starting;
			}
		}
		else
		{
			// play alternating tone and set status for victory
			m_statusText.setString("Game Over you Won");
			if (m_modeChangeTimer-- > 0)
			{
				if (m_modeChangeTimer % 50 == 0)
				{
					m_blueTone.play();
				}
				else
				{
					if (m_modeChangeTimer % 25 == 0)
					{
						m_redTone.play();
					}
				}
			}
			else
			{
				m_currentGameMdoe = GameMode::Starting;
			}
		}
		countdownTimers();
}
/// <summary>
/// @brief draw the window for the game.
/// 
/// draw buttons and text on left side
/// </summary>
```

```cpp
void Game::render()
{
	m_window.clear();
	m_window.draw(m_greenSquare);
	m_window.draw(m_redSquare);
	m_window.draw(m_yellowSquare);
	m_window.draw(m_blueSquare);
	m_window.draw(m_titleText);
	if (GameMode::Starting == m_currentGameMdoe )
	{
		m_window.draw(m_instructionsTextBlue);
		m_window.draw(m_instructionsTextGreen);
		m_window.draw(m_instructionsTextRed);
		m_window.draw(m_instructionsTextYellow);
	}
	m_window.draw(m_statusText);
	m_window.display();
}
```