Player.h

```
#pragma once
#include <SFML/Graphics.hpp>
using namespace sf;
class Player
{
private:
    const float START_SPEED = 200;
    const float START_HEALTH = 100;
    // Where is the player
    Vector2f m_Position;
    // The sprite
    Sprite m_Sprite;
    // And a texture
    // !!Watch this space – Changes here soon!!
    Texture m_Texture;
    // What is the screen resolution
    Vector2f m_Resolution;
    // What size is the current arena
    IntRect m_Arena;
    // How big is each tile of the arena
    int m_TileSize;
    // Which direction(s)the player is moving in
    bool m_UpPressed;
    bool m_DownPressed;
    bool m_LeftPressed;
    bool m_RightPressed;
    // How much health has the player got?
    int m_Health;
    // What is the max’  health the player can have
    int m_MaxHealth;
    // When was the player last hit
    Time m_LastHit;
    // Speed in pixels per second
    float m_Speed;
    // All our public functions next

public:
    Player();
    void spawn(IntRect arena, Vector2f resolution, int tileSize);
    // Call this at the end of every game
    void resetPlayerStats();
    // Handle the player getting hit by a zombie
    bool hit(Time timeHit);
    // How long ago was the player last hit
    Time getLastHitTime();
    // Where is the player
    FloatRect getPosition();
    // Where is the center of the player
    Vector2f getCenter();
    // What angle is the player facing
    float getRotation();
    // Send a copy of the sprite to the main function
    Sprite getSprite();
    // The next four functions move the player
    void moveLeft();
    void moveRight();
    void moveUp();
    void moveDown();
    // Stop the player moving in a specific direction
    void stopLeft();
    void stopRight();
    void stopUp();
    void stopDown();
    // We will call this function once every frame
    void update(float elapsedTime, Vector2i mousePosition);
    // Give the player a speed boost
    void upgradeSpeed();
    // Give the player some health
    void upgradeHealth();
    // Increase the max' health the player can have
    void increaseHealthLevel(int amount);
    // How much health has the player currently got?
    int getHealth();


};
```

Player.cpp

```
#include "Player.h"
#include <cmath>
Player::Player()
    : m_Speed(START_SPEED),
    m_Health(START_HEALTH),
    m_MaxHealth(START_HEALTH),
    m_Texture(),
    m_Sprite()
{
    // Associate a texture with the sprite
    // !!Watch this space!!
    m_Texture.loadFromFile("graphics/player.png");
    m_Sprite.setTexture(m_Texture);

    // Set the origin of the sprite to the center,
    // for smooth rotation
    m_Sprite.setOrigin(25, 25);
}

void Player::spawn(IntRect arena,
    Vector2f resolution,
    int tileSize)
{
    // Place the player in the middle of the arena
    m_Position.x = arena.width / 2;
    m_Position.y = arena.height / 2;
    // Copy the details of the arena
    // to the player's m_Arena
    m_Arena.left = arena.left;
    m_Arena.width = arena.width;
    m_Arena.top = arena.top;
    m_Arena.height = arena.height;
    // Remember how big the tiles are in this arena
    m_TileSize = tileSize;
    // Store the resolution for future use
    m_Resolution.x = resolution.x;
    m_Resolution.y = resolution.y;
}

void Player::resetPlayerStats()
{
    m_Speed = START_SPEED;
    m_Health = START_HEALTH;
    m_MaxHealth = START_HEALTH;
}

Time Player::getLastHitTime()
{
    return m_LastHit;
}
bool Player::hit(Time timeHit)
{
    if (timeHit.asMilliseconds()
        - m_LastHit.asMilliseconds() > 200)
    {
        m_LastHit = timeHit;
        m_Health -= 10;
        return true;
    }
    else
    {
        return false;
    }
}

FloatRect Player::getPosition(){
    return m_Sprite.getGlobalBounds();
}
Vector2f Player::getCenter(){
    return m_Position;
}
float Player::getRotation(){
    return m_Sprite.getRotation();
}
Sprite Player::getSprite(){
    return m_Sprite;
}
int Player::getHealth(){
    return m_Health;
}
void Player::moveLeft(){
    m_LeftPressed = true;
}
void Player::moveRight(){
    m_RightPressed = true;
}
```

```
void Player::moveUp()
{
   m_UpPressed = true;
}
void Player::moveDown()
{
   m_DownPressed = true;
}
void Player::stopLeft()
{
   m_LeftPressed = false;
}
void Player::stopRight()
{
   m_RightPressed = false;
}
void Player::stopUp()
{
   m_UpPressed = false;
}
void Player::stopDown()
{
   m_DownPressed = false;
}

void Player::update(float elapsedTime, Vector2i mousePosition)
{
   if (m_UpPressed)
   {
      m_Position.y -= m_Speed * elapsedTime;
   }
   if (m_DownPressed)
   {
      m_Position.y += m_Speed * elapsedTime;
   }
   if (m_RightPressed)
   {
      m_Position.x += m_Speed * elapsedTime;
   }
   if (m_LeftPressed)
   {
      m_Position.x -= m_Speed * elapsedTime;
   }
   m_Sprite.setPosition(m_Position);
```

```
    // Keep the player in the arena
    if (m_Position.x > m_Arena.width - m_TileSize)
    {
        m_Position.x = m_Arena.width - m_TileSize;
    }
    if (m_Position.x < m_Arena.left + m_TileSize)
    {
        m_Position.x = m_Arena.left + m_TileSize;
    }
    if (m_Position.y > m_Arena.height - m_TileSize)
    {
        m_Position.y = m_Arena.height - m_TileSize;
    }
    if (m_Position.y < m_Arena.top + m_TileSize)
    {
        m_Position.y = m_Arena.top + m_TileSize;
    }

    // Calculate the angle the player is facing
    float angle = (atan2(mousePosition.y - m_Resolution.y / 2,
        mousePosition.x - m_Resolution.x / 2)
        * 180) / 3.141;
    m_Sprite.setRotation(angle);
}

void Player::upgradeSpeed()
{
    // 20% speed upgrade
    m_Speed += (START_SPEED * .2);
}
void Player::upgradeHealth()
{
    // 20% max health upgrade
    m_MaxHealth += (START_HEALTH * .2);
}
void Player::increaseHealthLevel(int amount)
{
    m_Health += amount;
    // But not beyond the maximum
    if (m_Health > m_MaxHealth)
    {
        m_Health = m_MaxHealth;
    }
}
```

ZombieArena.h

```
#pragma once
#include <SFML/Graphics.hpp>
using namespace sf;

int createBackground(VertexArray& rVA, IntRect arena);
```

createBackground.cpp

```
#include "ZombieArena.h"
int createBackground(VertexArray& rVA, IntRect arena)
{
    // Anything we do to rVA we are really doing
    // to background (in the main function)

    // How big is each tile/texture
    const int TILE_SIZE = 50;
    const int TILE_TYPES = 3;
    const int VERTS_IN_QUAD = 4;

    int worldWidth = arena.width / TILE_SIZE;
    int worldHeight = arena.height / TILE_SIZE;
    // What type of primitive are we using?
    rVA.setPrimitiveType(Quads);
    // Set the size of the vertex array
    rVA.resize(worldWidth * worldHeight * VERTS_IN_QUAD);
    // Start at the beginning of the vertex array
    int currentVertex = 0;

    for (int w = 0; w < worldWidth; w++)
    {
        for (int h = 0; h < worldHeight; h++)
        {
            // Position each vertex in the current quad
            rVA[currentVertex + 0].position =
                Vector2f(w * TILE_SIZE, h * TILE_SIZE);

            rVA[currentVertex + 1].position =
                Vector2f((w * TILE_SIZE) + TILE_SIZE, h * TILE_SIZE);

            rVA[currentVertex + 2].position =
                Vector2f((w * TILE_SIZE) + TILE_SIZE, (h * TILE_SIZE)
                    + TILE_SIZE);

            rVA[currentVertex + 3].position =
                Vector2f((w * TILE_SIZE), (h * TILE_SIZE)
                    + TILE_SIZE);

            // Define the position in the Texture for current quad
            // Either grass, stone, bush or wall
            if (h == 0 || h == worldHeight - 1 ||
                w == 0 || w == worldWidth - 1)
            {
                // Use the wall texture
                rVA[currentVertex + 0].texCoords =
                    Vector2f(0, 0 + TILE_TYPES * TILE_SIZE);

                rVA[currentVertex + 1].texCoords =
                    Vector2f(TILE_SIZE, 0 +
                        TILE_TYPES * TILE_SIZE);

                rVA[currentVertex + 2].texCoords =
                    Vector2f(TILE_SIZE, TILE_SIZE +
                        TILE_TYPES * TILE_SIZE);

                rVA[currentVertex + 3].texCoords =
                    Vector2f(0, TILE_SIZE +
                        TILE_TYPES * TILE_SIZE);
            }
            else
            {
                // Use a random floor texture
                srand((int)time(0) + h * w - h);
                int mOrG = (rand() % TILE_TYPES);
                int verticalOffset = mOrG * TILE_SIZE;
                rVA[currentVertex + 0].texCoords =
                    Vector2f(0, 0 + verticalOffset);

                rVA[currentVertex + 1].texCoords =
                    Vector2f(TILE_SIZE, 0 + verticalOffset);

                rVA[currentVertex + 2].texCoords =
                    Vector2f(TILE_SIZE, TILE_SIZE + verticalOffset);

                rVA[currentVertex + 3].texCoords =
                    Vector2f(0, TILE_SIZE + verticalOffset);
            }


            // Position ready for the next four vertices
            currentVertex = currentVertex + VERTS_IN_QUAD;
        }
    }

    return TILE_SIZE;
}
```

ZombieArena.cpp

```
#include "Player.cpp"
#include "CreateBackground.cpp"

int main()
{
   // The game will always be in one of four states
   enum class State {
      PAUSED, LEVELING_UP,
      GAME_OVER, PLAYING
   };

   // Start with the GAME_OVER state
   State state = State::GAME_OVER;
   // Get the screen resolution and
   // create an SFML window
   Vector2f resolution;
   resolution.x =
      VideoMode::getDesktopMode().width;
   resolution.y =
      VideoMode::getDesktopMode().height;
   RenderWindow window(
      VideoMode(resolution.x, resolution.y),
      "Zombie Arena", Style::Fullscreen);
   // Create a an SFML View for the main action
   View mainView(sf::FloatRect(0, 0,
      resolution.x, resolution.y));
   // Here is our clock for timing everything
   Clock clock;
   // How long has the PLAYING state been active
   Time gameTimeTotal;
   // Where is the mouse in
   // relation to world coordinates
   Vector2f mouseWorldPosition;
   // Where is the mouse in
   // relation to screen coordinates
   Vector2i mouseScreenPosition;
   // Create an instance of the Player class
   Player player;
   // The boundaries of the arena
   IntRect arena;
   // Create the background
   VertexArray background;
```

```
// Load the texture for our background vertex array
Texture textureBackground;
textureBackground.loadFromFile("graphics/background_sheet.png");

// The main game loop
while (window.isOpen())
{
    /*
************
Handle input
************
*/
// Handle events by polling
    Event event;
    while (window.pollEvent(event))
    {
        if (event.type == Event::KeyPressed)
        {
            // Pause a game while playing
            if (event.key.code == Keyboard::Return &&
                state == State::PLAYING)
            {
                state = State::PAUSED;
            }
            // Restart while paused
            else if (event.key.code == Keyboard::Return &&
                state == State::PAUSED)
            {
                state = State::PLAYING;
                // Reset the clock so there isn't a frame jump
                clock.restart();
            }
            // Start a new game while in GAME_OVER state
            else if (event.key.code == Keyboard::Return &&
                state == State::GAME_OVER)
            {
                state = State::LEVELING_UP;
            }
            if (state == State::PLAYING)
            {
            }
        }
    }// End event polling
```

```
// Handle the player quitting
if (Keyboard::isKeyPressed(Keyboard::Escape))
{
    window.close();
}

// Handle WASD while playing
if (state == State::PLAYING)
{
    // Handle the pressing and releasing of WASD keys
    if (Keyboard::isKeyPressed(Keyboard::W))
    {
        player.moveUp();
    }
    else
    {
        player.stopUp();
    }
    if (Keyboard::isKeyPressed(Keyboard::S))
    {
        player.moveDown();
    }
    else
    {
        player.stopDown();
    }
    if (Keyboard::isKeyPressed(Keyboard::A))
    {
        player.moveLeft();
    }
    else
    {
        player.stopLeft();
    }
    if (Keyboard::isKeyPressed(Keyboard::D))
    {
        player.moveRight();
    }
    else
    {
        player.stopRight();
    }
}// End WASD while playing
```

```
// Handle the LEVELING up state
if (state == State::LEVELING_UP)
{
   // Handle the player LEVELING up
   if (event.key.code == Keyboard::Num1)
   {
      state = State::PLAYING;
   }
   if (event.key.code == Keyboard::Num2)
   {
      state = State::PLAYING;
   }
   if (event.key.code == Keyboard::Num3)
   {
      state = State::PLAYING;
   }
   if (event.key.code == Keyboard::Num4)
   {
      state = State::PLAYING;
   }
   if (event.key.code == Keyboard::Num5)
   {
      state = State::PLAYING;
   }
   if (event.key.code == Keyboard::Num6)
   {
      state = State::PLAYING;
   }

   if (state == State::PLAYING)
   {
      // Prepare the level
      // We will modify the next two lines later
      arena.width = 500;
      arena.height = 500;
      arena.left = 0;
      arena.top = 0;
      // Pass the vertex array by reference
      // to the createBackground function
      int tileSize = createBackground(background, arena);
      // We will modify this line of code later
      // int tileSize = 50;

      // Spawn the player in middle of the arena
```

```
            player.spawn(arena, resolution, tileSize);

            // Reset clock so there isn't a frame jump
            clock.restart();
        }
    }// End LEVELING up

    /*
****************
UPDATE THE FRAME
****************
*/
    if (state == State::PLAYING)
    {
        // Update the delta time
        Time dt = clock.restart();

        // Update the total game time
        gameTimeTotal += dt;

        // Make a fraction of 1 from the delta time
        float dtAsSeconds = dt.asSeconds();
        // Where is the mouse pointer
        mouseScreenPosition = Mouse::getPosition();
        // Convert mouse position to world
        // based coordinates of mainView
        mouseWorldPosition = window.mapPixelToCoords(
            Mouse::getPosition(), mainView);
        // Update the player
        player.update(dtAsSeconds, Mouse::getPosition());
        // Make a note of the players new position
        Vector2f playerPosition(player.getCenter());

        // Make the view centre
        // the around player
        mainView.setCenter(player.getCenter());
    }// End updating the scene


/*
**************
Draw the scene
**************
*/
```

```
        if (state == State::PLAYING)
        {
            window.clear();
            // set the mainView to be displayed in the window
            // And draw everything related to it
            window.setView(mainView);

            // Draw the background
            window.draw(background, &textureBackground);

            // Draw the player
            window.draw(player.getSprite());
        }

        if (state == State::LEVELING_UP)
        {
        }

        if (state == State::PAUSED)
        {
        }

        if (state == State::GAME_OVER)
        {
        }

        window.display();

    }// End game loop

    return 0;

}
```