Motion Graphics

Dr Noel O’Hara

Contents

[First Simple SFML 2](#_Toc158796027)

[How to represent the array of projectiles. 6](#_Toc158796028)

[How the Player control the projectiles 7](#_Toc158796029)

[Collision 8](#_Toc158796030)

[How Create a Starfield Effect 13](#_Toc158796031)

[Particle Systems 18](#_Toc158796032)

[Physics 24](#_Toc158796033)

[Player Jumping using Physics 28](#_Toc158796034)

[Walking/Running Movement using Physics 30](#_Toc158796035)

[Platform Game (One Button) 35](#_Toc158796036)

# First Simple SFML

//Noel O' Hara

//first simple game.

#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-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-window.lib")

#pragma comment(lib,"sfml-network.lib")

#endif

#include <SFML/Graphics.hpp>

#include <iostream>

#include <stdlib.h>

#include <time.h>

int main()

{

sf::RenderWindow window(sf::VideoMode(800, 600), "First Graphics in C++");

sf::RectangleShape simpleRectangle;

float width = 30;

float height = 30;

simpleRectangle.setSize(sf::Vector2f(width, height));

simpleRectangle.setFillColor(sf::Color::Red);

simpleRectangle.setPosition(100, 200);

srand(time(NULL));

const int numCircles = 10;

sf::CircleShape circles[numCircles];

for (int index = 0; index < numCircles; index++)

{

circles[index].setFillColor(sf::Color::White);

circles[index].setRadius(10);

circles[index].setPosition(40 \* index, 200);

}

float xPosition = rand() % 800;

float yPosition = 200;

float speed = 1;

bool spacePressed = false;

sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);

sf::Time timeSinceLastUpdate = sf::Time::Zero;

sf::Clock clockForFrameRate;

clockForFrameRate.restart();

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

}

timeSinceLastUpdate += clockForFrameRate.restart();

if (timeSinceLastUpdate > timePerFrame)

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))

{

if (spacePressed==false)

{

speed \*= -1;

spacePressed = true;

}

}

else {

spacePressed = false;

}

xPosition += speed;

simpleRectangle.setPosition(xPosition, yPosition);

for (int index = 0; index < numCircles; index++)

{

if (simpleRectangle.getGlobalBounds().intersects(circles[index].getGlobalBounds()))

{

circles[index].setPosition(1000, 1000);

}

}

window.clear();

for (int index = 0; index < numCircles; index++)

{

window.draw(circles[index]);

}

window.draw(simpleRectangle);

window.display();

timeSinceLastUpdate = sf::Time::Zero;

}

}

return 0;

}

# How to represent the array of projectiles.

Initialize

We will just use an array sf::RectangleShape

|  |
| --- |
| sf::Vector2f offScreenPosition{ 20000,20000 };  float projectileSpeed = -5;  const int numProjectiles = 10;  sf::RectangleShape projectiles[numProjectiles];  At the start initialize the rectangleShapes, size ,color and setPosition to **be off the screen**  for (int i = 0; i < numProjectiles; i++)  {  projectiles[i].setSize(sf::Vector2f(20, 20));  projectiles[i].setPosition(offScreenPosition);  projectiles[i].setFillColor(sf::Color::Yellow);  } |

Updating the projectiles

Only want to update projectiles that are not in the offscreen position

If its not in the offscreen position

Move it up

If off the top of the screen move it to offscreen position

|  |
| --- |
| for (int i = 0; i < numProjectiles; i++)  {  if (projectiles[i].getPosition().x != offScreenPosition.x)  {  projectiles[i].move(0, projectileSpeed);  if (projectiles[i].getPosition().y < 0)  {  projectiles[i].setPosition(offScreenPosition);  }  }  } |

Drawing the projectiles

|  |
| --- |
| for (int i = 0; i < numProjectiles; i++)  {  window.draw(projectiles[i]);  } |

# How the Player control the projectiles

Logic:

If player presses space

Loops through the projectile array

If the projectile is offscreen

Put the position of the projectile to the player positions

Exit loop

It’s a little more complicated than the Above

|  |
| --- |
| bool readyToFire = true;  int waitToFireCounter = 10;  Int waitToFireInterval = waitToFireCounter;  …  …  if (readyToFire == true)  {  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  {  for (int i = 0; i < numProjectiles; i++)  {  if (projectiles[i].getPosition()== offScreenPosition)  {  projectiles[i].setPosition(player.getPosition());  readyToFire = false;  waitToFireCounter = waitToFireInterval;  break;  }  }  }  }  else {  waitToFireCounter--;  if (waitToFireCounter <= 0)  {  readyToFire = true;  }  } |

## Collision

Loop through each projectile

Loop through each enemy

Check if enemy collides with projectile

Add to score

Move projectile offscreen

Respawn enemy to the top

|  |
| --- |
| for (int npcIndex = 0; npcIndex < numberOfNPCs; npcIndex++)  {  NonPlayerSprites[npcIndex].move(0, enemySpeed);  if (NonPlayerSprites[npcIndex].getPosition().y > window.getSize().y)  {  score = score + 1;  if (enemySpeed < maxEnemySpeed)  {  enemySpeed = enemySpeed + 1;  }  randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[npcIndex].getGlobalBounds().width);  NonPlayerSprites[npcIndex].setPosition(randomRespawnX, 0);  }  for (int projectileIndex = 0; projectileIndex < numProjectiles; projectileIndex++)  {  if (NonPlayerSprites[npcIndex].getGlobalBounds().intersects(projectiles[projectileIndex].getGlobalBounds()))  {  projectiles[projectileIndex].setPosition(offScreenPos);  randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[npcIndex].getGlobalBounds().width);  NonPlayerSprites[npcIndex].setPosition(randomRespawnX, 0);  }  }  } |
|  |
|  |

25 NPC’s

10 projectiles

Line below is executed 250 times per frame.

if (NonPlayerSprites[npcIndex].getGlobalBounds().intersects(projectiles[projectileIndex].getGlobalBounds()))

Full Code

|  |
| --- |
| // Game with arrays of array of projectiles , sprites text, shapes, sprites and collision    // Author Noel O'Hara  // |
| #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-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-window.lib")  #pragma comment(lib,"sfml-network.lib")  #endif    #include <SFML/Graphics.hpp>  #include <iostream>  #include <stdlib.h>  #include <time.h>    int main()  {          sf::RenderWindow window(sf::VideoMode(800, 600), "sprites, text ,projectiles, player and collisions");    srand(time(NULL));        bool readyToFire = true;  int waitToFireInterval = 10;  int waitToFireCounter = waitToFireInterval;  sf::Vector2f playerVelocity{ 0,0 };  sf::Vector2f offScreenPos{ -2000,-2000 };  const int numProjectiles = 10;    sf::RectangleShape projectiles[numProjectiles];  for (int i = 0; i < numProjectiles; i++)  {  projectiles[i].setSize(sf::Vector2f(20, 20));  projectiles[i].setPosition(offScreenPos);  projectiles[i].setFillColor(sf::Color::Yellow);  }  float projectileSpeed = -5;  float enemySpeed = 2;  float maxEnemySpeed = 5;  sf::Font font;  if (!font.loadFromFile("arial.ttf"))  {  // error...  }  int score = 0;  sf::Text scoreText;  scoreText.setFont(font);    scoreText.setString("Score: " + std::to\_string(score));  scoreText.setCharacterSize(24);  scoreText.setFillColor(sf::Color::White);        sf::Texture npcTexture;  if (!npcTexture.loadFromFile("Mace.png"))  {  // error...  }  const int numberOfNPCs = 20;  sf::Sprite NonPlayerSprites[numberOfNPCs];    float randomRespawnX;  for (int i = 0; i < numberOfNPCs; i++)  {  NonPlayerSprites[i].setTexture(npcTexture);    randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[i].getGlobalBounds().width);  NonPlayerSprites[i].setPosition(randomRespawnX, 0);  }        sf::CircleShape player(30);  player.setFillColor(sf::Color::Red);  player.setPosition(200, window.getSize().y - player.getGlobalBounds().height);  float playerSpeed = 5;          // set timePerFrame to 1 60th of a second. 60 frames per second  sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);  sf::Time timeSinceLastUpdate = sf::Time::Zero;  // the clock object keeps the time.  sf::Clock clock;  clock.restart();    while (window.isOpen())  {  // check if the close window button is clicked on.  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }    //add to the time since last update and restart the clock  timeSinceLastUpdate += clock.restart();    //update every 60th of a second  //only when the time since last update is greater than 1/60 update the world.    if (timeSinceLastUpdate > timePerFrame)  {      // get keyboard input.  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))  {  player.move(-playerSpeed, 0);      }  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))  {  player.move(playerSpeed, 0);    }        if (waitToFireCounter == 0)  {  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  {  for (int i = 0; i < numProjectiles; i++)  {  if (projectiles[i].getPosition().x == offScreenPos.x)  {  projectiles[i].setPosition(player.getPosition());    waitToFireCounter = waitToFireInterval;  break;    }    }  }    }  else {  waitToFireCounter--;    }        if (player.getPosition().x < 0)  {  player.setPosition(window.getSize().x - player.getGlobalBounds().width, player.getPosition().y);  }  for (int i = 0; i < numProjectiles; i++)  {  if (projectiles[i].getPosition().x != offScreenPos.x)  {  projectiles[i].move(0, projectileSpeed);  if (projectiles[i].getPosition().y < 0)  {  projectiles[i].setPosition(offScreenPos);  }  }  }  for (int npcIndex = 0; npcIndex < numberOfNPCs; npcIndex++)  {  NonPlayerSprites[npcIndex].move(0, enemySpeed);  if (NonPlayerSprites[npcIndex].getPosition().y > window.getSize().y)  {  score = score + 1;  if (enemySpeed < maxEnemySpeed)  {  enemySpeed = enemySpeed + 1;  }  randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[npcIndex].getGlobalBounds().width);  NonPlayerSprites[npcIndex].setPosition(randomRespawnX, 0);  }  for (int projectileIndex = 0; projectileIndex < numProjectiles; projectileIndex++)  {  if (NonPlayerSprites[npcIndex].getGlobalBounds().intersects(projectiles[projectileIndex].getGlobalBounds()))  {  projectiles[projectileIndex].setPosition(offScreenPos);  randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[npcIndex].getGlobalBounds().width);  NonPlayerSprites[npcIndex].setPosition(randomRespawnX, 0);    }  }    }      for (int i = 0; i < numberOfNPCs; i++)  {  if (NonPlayerSprites[i].getGlobalBounds().intersects(player.getGlobalBounds()))  {  score = score - 1;  randomRespawnX = rand() % int(window.getSize().x - NonPlayerSprites[i].getGlobalBounds().width);  NonPlayerSprites[i].setPosition(randomRespawnX, 0);  }  }    scoreText.setString("Score: " + std::to\_string(score));    window.clear();    window.draw(player);  for (int i = 0; i < numberOfNPCs; i++)  {  window.draw(NonPlayerSprites[i]);  }  for (int i = 0; i < numProjectiles; i++)  {  window.draw(projectiles[i]);  }    window.draw(scoreText);    window.display();    // reset the timeSinceLastUpdate to 0  timeSinceLastUpdate = sf::Time::Zero;  }  }    return 0;  } |

# How Create a Starfield Effect

Stars move down the screen at different speeds to simulate a starfield as

a background.

A picture containing outdoor object, star, dark, night sky

Description automatically generated

First attempt:

Randomly place stars on the screen and then move them down at the same speed.

When they reach the bottom of the screen place them randomly again.

Class StarField that contains an array of Stars.

Init()

Update()

Draw()

class Star

{

public:

sf::RectangleShape shape;

sf::Vector2f velocity;

};

class Starfield

{

public:

static const int numStars = 50;

Star stars[numStars];

void Init(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

//set the shape size

stars[i].shape.setSize(sf::Vector2f(3, 3));

stars[i].shape.setFillColor(sf::Color::White);

//set the velocity, all the same velocity

stars[i].velocity = { 0,10 };

//set starting position of the star

float startingX = rand() % int(win.getSize().x);

float startingY = rand() % int(win.getSize().y);

stars[i].shape.setPosition(startingX, startingY);

}

}

Starfield()

{

}

void Draw(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

win.draw(stars[i].shape);

}

}

void Update(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

stars[i].shape.move(stars[i].velocity);

if (stars[i].shape.getPosition().y > win.getSize().y)

{

float startingX = rand() % int(win.getSize().x);

float startingY = rand() % int(win.getSize().y);

stars[i].shape.setPosition(startingX, startingY);

}

}

}

};

// to initialise it

Starfield starField();

starField.Init(window);

//To draw it

starField.Draw(window);

//to Update it

starField.Update(window);

it doesn’t really look like a starfield yet..

Changes:

In Init()

//move the star above the window instead in the window

stars[i].shape.move(0, -int(win.getSize().y));

//set a random velocity

stars[i].velocity = { 0,float(rand()% 10) };

and in Update()

stars[i].shape.move(0, -int(win.getSize().y));

Put them both in Starfield.h

class Star

{

public:

sf::RectangleShape shape;

sf::Vector2f velocity;

};

class Starfield

{

public:

static const int numStars = 50;

Star stars[numStars];

void Init(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

//set the shape size

stars[i].shape.setSize(sf::Vector2f(3, 3));

stars[i].shape.setFillColor(sf::Color::White);

//set the velocity

stars[i].velocity = { 0,float(rand()% 10) };

//set starting position of the star

float startingX = rand() % int(win.getSize().x);

float startingY = rand() % int(win.getSize().y);

stars[i].shape.setPosition(startingX, startingY);

//move the star above the window

stars[i].shape.move(0, -int(win.getSize().y));

}

}

Starfield()

{

}

void Draw(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

win.draw(stars[i].shape);

}

}

void Update(sf::RenderWindow& win)

{

for (int i = 0; i < numStars; i++)

{

stars[i].shape.move(stars[i].velocity);

if (stars[i].shape.getPosition().y > win.getSize().y)

{

float startingX = rand() % int(win.getSize().x);

float startingY = rand() % int(win.getSize().y);

stars[i].shape.setPosition(startingX, startingY);

//move the star above the window

stars[i].shape.move(0, -int(win.getSize().y));

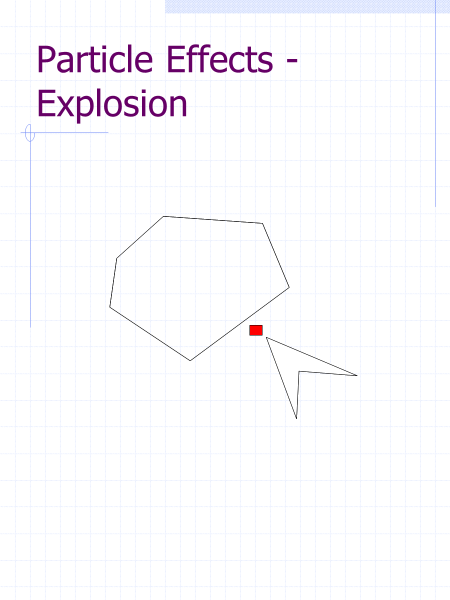
}

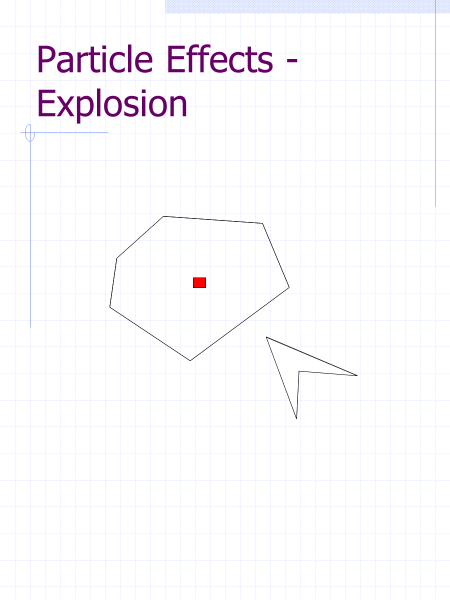
}

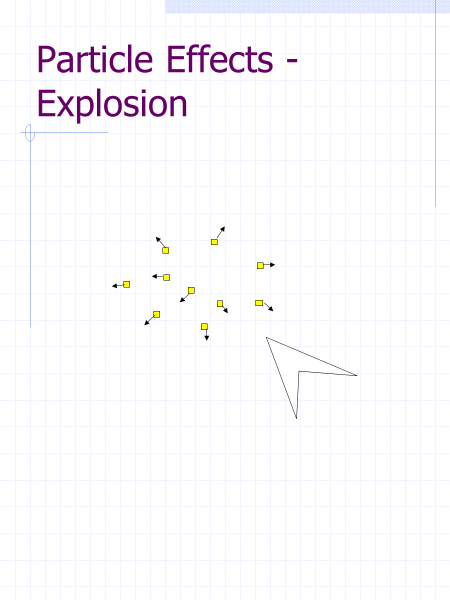
}

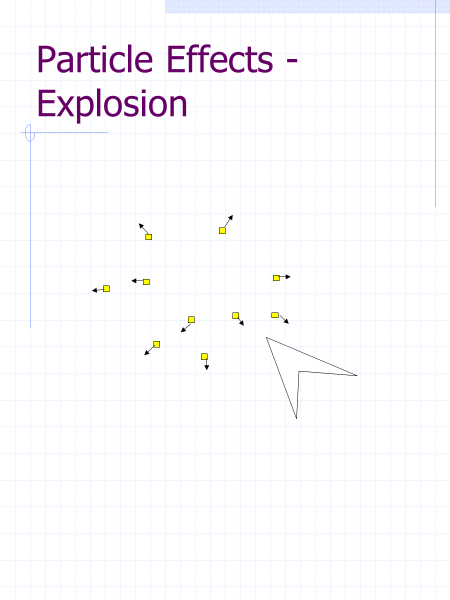
};

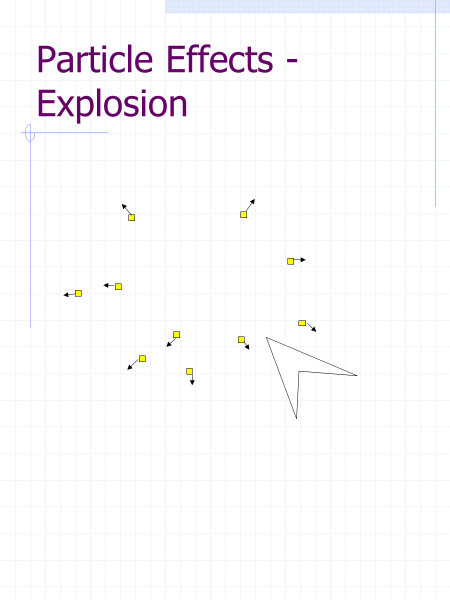
# Particle Systems











First create a Particle Class

|  |
| --- |
| class Particle  {  public:  int timetoLive;  sf::Vector2f velocity;  sf::RectangleShape shape;  void Draw(sf::RenderWindow& win)  {  if (timetoLive > 0)  {  win.draw(shape);  }  }  void Update()  {  if (timetoLive > 0)  {  shape.move(velocity);  timetoLive--;  }  }  Particle() {}  Particle(sf::Vector2f pos, sf::Vector2f vel)  {  shape.setSize(sf::Vector2f(6, 6));  shape.setPosition(pos);  shape.setFillColor(sf::Color::Blue);  velocity = vel;  //timetoLive = 200;  timetoLive = rand() % 150;  }  }; |

ParticleSystem

|  |
| --- |
| #define maxParticles 50  class ParticleSystem  {  public:    Particle particles[maxParticles];  sf::Vector2f position;  void Initialise(sf::Vector2f pos)  {  position=pos;  for (int i =0;i<maxParticles;i++)  {  particles[i] = Particle(position,sf::Vector2f(rand() / double(RAND\_MAX)\*4-2,rand() / double(RAND\_MAX)\*4-2));  }  }  void Update()  {  for (int i =0;i<maxParticles;i++)  {    particles[i].Update();      }  }  void Draw(sf::RenderWindow &win )  {  for (int i =0;i<maxParticles;i++)  {  particles[i].Draw(win);  }  }  ParticleSystem(){}  }; |

Creating the velocities

We need to set Random velocities around 360 degree.

In the example below we create a random velocity in x between -2 and 2 and in y between -2 and 2.

The code below return a floating point value between 0 and 1

rand() / double(RAND\_MAX)

for (int i =0;i<maxParticles;i++)

{

particles[i] = Particle(position,sf::Vector2f(rand() / double(RAND\_MAX)\*4-2,rand() / double(RAND\_MAX)\*4-2));

}

# Physics

Starter code to simulate gravity. (Below)

We will start by simulating a falling circle under gravity.

|  |
| --- |
| //First physics  // Noel O’Hara  #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-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-window.lib")  #pragma comment(lib,"sfml-network.lib")  #endif  #include <SFML/Graphics.hpp>  int main()  {  sf::RenderWindow window(sf::VideoMode(800, 800), "Go Physics!!");  sf::CircleShape shape(0.5f);  shape.setFillColor(sf::Color::Green);  sf::Vector2f velocity(0, 0);  sf::Vector2f position(400, 400);  sf::Vector2f gravity(0.0f, 9.8f);  shape.setPosition(position);  sf::Clock clock;  const float FPS = 60.0f;  const sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);  sf::Time timeSinceLastUpdate = sf::Time::Zero;  clock.restart();  while (window.isOpen())  {  //read keyboard inout  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }  //get the time since last update and restart the clock  timeSinceLastUpdate += clock.restart();  //update every 60th of a second  if (timeSinceLastUpdate > timePerFrame)  {  window.clear();  position = shape.getPosition();  float timeChange = (float)timeSinceLastUpdate.asSeconds();    //update position and velocity    shape.setPosition(position);  window.draw(shape);  window.display();  timeSinceLastUpdate = sf::Time::Zero;  }  }  return 0;  } |
|  |

The above code draws a circle at 400,400.

How do we simulate.

We use velocity to update the position.

**position = position+ velocity;**

According to Newtonian physics what is gravity with respect to velocity?

Fill in the blank .

\_\_\_\_\_\_ due to gravity is 9.8mss [answer Acceleration] .

How to change velocity with acceleration.?

**velocity = velocity + acceleration.**

We also have to add time into the calculation.

**position = position+ velocity \* timeChange;**

velocity = velocity + acceleration\* timeChange;

in our example gravity is acceleration

**velocity = velocity + gravity\* timeChange;**

Add the 2 lines above in bold beneath “//update position and velocity” in the code above.

In our simulation 1 pixel is a meter!

The make the physics simulation work in your game scale we introduce a pixelToMeter floating point number.

We scale gravity by that amount.

|  |
| --- |
| float pixelsPerMeter = 50;  sf::Vector2f gravity(0.0f, 9.8f\* pixelsPerMeter); |

Alter the pixelsPerMeter to have stronger or weaker in your game.

Bouncing off ground

When the bottom of the ball hits the bottom of the screen.

Negate the velocity in the Y direction.

|  |
| --- |
| if (position.y + shape.getGlobalBounds().height >= window.getSize().y)  {  velocity.y = -velocity.y;  } |

Losing energy when bouncing off a surface. It’s called the coefficient of restitution.

Multiply y velocity by a value between 0 and 1.

|  |
| --- |
| if (position.y + shape.getGlobalBounds().height >= window.getSize().y)  {  velocity.y = -velocity.y;  velocity.y \*= .6;    } |
|  |

Once the shape hits the ground we reposition the shapes y so that the bottom of the shape is exactly on the ground.

|  |
| --- |
| if (position.y + shape.getGlobalBounds().height >= window.getSize().y)  {  position.y = window.getSize().y - shape.getGlobalBounds().height;  velocity.y = -velocity.y;  velocity.y \*= .6;  } |

# Player Jumping using Physics

Initialise the Position of the circleShape to the bottom on the screen.

When space the player jumps under gravity.

Make the player jump by setting its y velocity to a negative value.

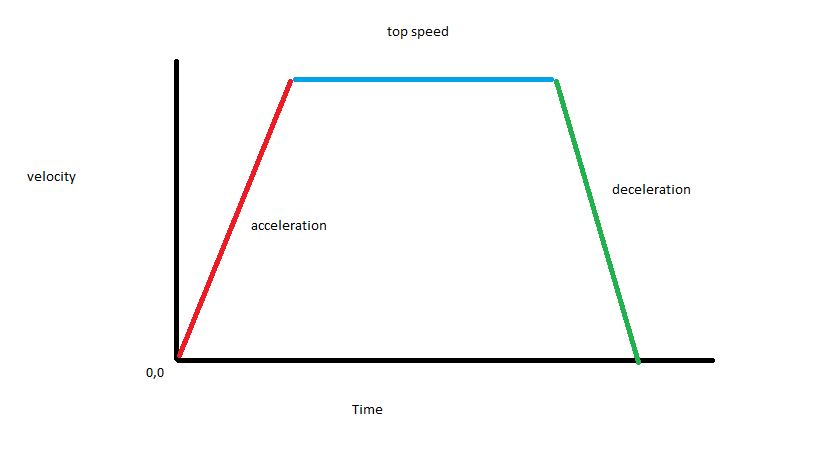
|  |
| --- |
| // player jumping  // Author Noel O' Hara  #ifdef \_DEBUG  #pragma comment(lib,"sfml-graphics-d.lib")  #pragma comment(lib,"sfml-audio-d.lib")  #pragma comment(lib,"sfml-system-d.lib")  #pragma comment(lib,"sfml-window-d.lib")  #pragma comment(lib,"sfml-network-d.lib")  #else  #pragma comment(lib,"sfml-graphics.lib")  #pragma comment(lib,"sfml-audio.lib")  #pragma comment(lib,"sfml-system.lib")  #pragma comment(lib,"sfml-window.lib")  #pragma comment(lib,"sfml-network.lib")  #endif  #include <SFML/Graphics.hpp>  int main()  {  sf::RenderWindow window(sf::VideoMode(800, 800), "Go Physics!!");  sf::CircleShape playerShape(25.0f);  playerShape.setFillColor(sf::Color::Green);  sf::Vector2f velocity(0, 0);    sf::Vector2f position(100, window.getSize().y - playerShape.getRadius() \*2);  float pixelsPerMeter = 100;  sf::Vector2f pushOff(0, -10 \* pixelsPerMeter);  sf::Vector2f gravity(0.0f, 9.8f\* pixelsPerMeter);      sf::Clock clock;  const float FPS = 60.0f;  const sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);  sf::Time timeSinceLastUpdate = sf::Time::Zero;  clock.restart();  while (window.isOpen())  {  //read keyboard inout  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }  //get the time since last update and restart the clock  timeSinceLastUpdate += clock.restart();  //update every 60th of a second  if (timeSinceLastUpdate > timePerFrame)  {  window.clear();  float timeChange = (float)timeSinceLastUpdate.asSeconds();  // if the player is jumping  if (position.y + playerShape.getGlobalBounds().height < window.getSize().y)  {  //update position and velocity  position = position + velocity \* timeChange;  velocity = velocity + gravity \* timeChange;  }  // the player is on the ground  else {  //reposition the player to be on the ground  //set y velocity to 0.  position.y = window.getSize().y - playerShape.getGlobalBounds().height;  velocity.y = 0;  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  {  velocity.y = pushOff.y;  position = position + velocity \* timeChange;  }  }    playerShape.setPosition(position);  window.draw(playerShape);  window.display();  timeSinceLastUpdate = sf::Time::Zero;  }  }  return 0;  } |

## Walking/Running Movement using Physics

Up to now we have moved a character just adding or subtracting a speed variable to the position when a key is pressed.

Instantaneously reached top speed from standing start.

We want to have an acceleration until we hit top speed and a deceleration when we let go the key



|  |
| --- |
| // player jumping and running  // Author Noel O' Hara  #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-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-window.lib")  #pragma comment(lib,"sfml-network.lib")  #endif  #include <SFML/Graphics.hpp>  #include "Trigonometry.hpp"  #include "VectorAlgebra2D.h"  int main()  {  sf::RenderWindow window(sf::VideoMode(800, 800), "Go Physics!!");  sf::RectangleShape playerShape(sf::Vector2f(50.0f,50.0f));  playerShape.setFillColor(sf::Color::Green);  sf::Vector2f velocity(0, 0);  sf::Vector2f position(100, window.getSize().y - playerShape.getGlobalBounds().height);  float pixelsPerMeter = 200;  sf::Vector2f jumpPushOff(0, -3 \* pixelsPerMeter);  sf::Vector2f gravity(0.0f, 9.8f \* pixelsPerMeter);  float coeffFriction= 0.4;  sf::Vector2f unitVelocity;  sf::Vector2f xDecceleration = -coeffFriction \* gravity.y \* unitVelocity;  sf::Vector2f xGroundedAcceleration = { 1\* pixelsPerMeter,0 };  sf::Vector2f xJumpingAcceleration = { 0.5f \* pixelsPerMeter,0 };  float maxGroundedXVelocity = 2 \* pixelsPerMeter;  float maxMidAirXVelocity = 1 \* pixelsPerMeter;  sf::Clock clock;  const float FPS = 60.0f;  const sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);  sf::Time timeSinceLastUpdate = sf::Time::Zero;  clock.restart();  while (window.isOpen())  {  //read keyboard inout  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }  //get the time since last update and restart the clock  timeSinceLastUpdate += clock.restart();  //update every 60th of a second  if (timeSinceLastUpdate > timePerFrame)  {  window.clear();  float timeChange = (float)timeSinceLastUpdate.asSeconds();  // if the player is jumping  if (position.y + playerShape.getGlobalBounds().height < window.getSize().y)  {  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))  {  if (velocity.x < maxMidAirXVelocity)  {  velocity += xJumpingAcceleration;  }    }  else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))  {  if (velocity.x > -maxMidAirXVelocity)  {  velocity -= xJumpingAcceleration;  }    }  //update position and velocity  position = position + velocity \* timeChange;  velocity = velocity + gravity \* timeChange;  }  // the player is on the ground  else {    position.y = window.getSize().y - playerShape.getGlobalBounds().height;  velocity.y = 0;  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  {  velocity.y = jumpPushOff.y;  position = position + velocity \* timeChange;  }    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))  {    if (velocity.x < maxGroundedXVelocity)  {  velocity += xGroundedAcceleration;    }  position = position + velocity \* timeChange;  }  else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))  {  if (velocity.x > -maxGroundedXVelocity)  {  velocity -= xGroundedAcceleration;  }  position = position + velocity \* timeChange;  }  else {    if (velocity.x != 0)  {  unitVelocity = thor::unitVector(velocity);  xDecceleration = -coeffFriction \* gravity.y \* unitVelocity;  position = position + velocity \* timeChange;  velocity = velocity + xDecceleration \* timeChange;  }  }  }  playerShape.setPosition(position);  window.draw(playerShape);  window.display();  timeSinceLastUpdate = sf::Time::Zero;  }  }  return 0;  } |

Acceleration

Accelerating to top speed.

Below is the code that is executed when the player presses the right key.

|  |
| --- |
| if (velocity.x < maxGroundedXVelocity)  {  velocity += xGroundedAcceleration;    }  position = position + velocity \* timeChange; |

Deceleration

Physics formula for friction

coeffFriction between 0 and 1

the closer coeffFriction gets to 1 more friction is simulated

xDecceleration = -coeffFriction \* gravity.y \* unitVelocity;

UnitVelocity is calculated using

unitVelocity = thor::unitVector(velocity);

## Platform Game (One Button)

|  |
| --- |
| // platform game.....  // Author :Noel O' Hara  #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-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-window.lib")  #pragma comment(lib,"sfml-network.lib")  #endif  #include <SFML/Graphics.hpp>  #include <iostream>  #include <time.h>  class Game  {  public:  //create Window  sf::RenderWindow window;  sf::View view;  float randomNum;  sf::RectangleShape playerShape;  float velocityX = 0, velocityY = 0, gravity = 0.3;  static const int numRows = 45;  static const int numCols = 20;  int levelData[numRows][numCols] =  {  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,2,2 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,2,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,1,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0 },  { 0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0 },  { 0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0 },  { 0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,0,0 },  { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 },  { 0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0 } };  sf::RectangleShape level[numRows][numCols];  Game()  {  window.create(sf::VideoMode(800, 600), "Endless Runner Game");  }  void init()  {  view = window.getDefaultView();  playerShape.setSize(sf::Vector2f(20, 20));  playerShape.setPosition(160, 500);  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  if (levelData[row][col] == 1)  {  level[row][col].setSize(sf::Vector2f(70, 30));  level[row][col].setPosition(row \* 70, col \* 30);  level[row][col].setFillColor(sf::Color::Red);  }  if (levelData[row][col] == 0)  {  level[row][col].setSize(sf::Vector2f(70, 30));  level[row][col].setPosition(row \* 70, col \* 30);  level[row][col].setFillColor(sf::Color::Black);  }  if (levelData[row][col] == 2)  {  level[row][col].setSize(sf::Vector2f(70, 30));  level[row][col].setPosition(row \* 70, col \* 30);  level[row][col].setFillColor(sf::Color::Blue);  }  }  std::cout << std::endl;  }  }  void run()  {  sf::Time timePerFrame = sf::seconds(1.0f / 60.0f);  sf::Time timeSinceLastUpdate = sf::Time::Zero;  sf::Clock clock;  clock.restart();  while (window.isOpen())  {  sf::Event event;  while (window.pollEvent(event))  {  if (event.type == sf::Event::Closed)  window.close();  }  timeSinceLastUpdate += clock.restart();  if (timeSinceLastUpdate > timePerFrame)  {  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  level[row][col].move(-3.7, 0);  }  }  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space) && velocityY == 0)  {  velocityY = -11.8;  }  velocityY = velocityY + gravity;  playerShape.move(0, velocityY);  gravity = 0.6;  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  if (velocityY >= 0)  {  if (levelData[row][col] == 1)  {  if (playerShape.getGlobalBounds().intersects(level[row][col].getGlobalBounds()))  {  if (playerShape.getPosition().y < level[row][col].getPosition().y)  {  gravity = 0;  velocityY = 0;  playerShape.setPosition(playerShape.getPosition().x, level[row][col].getPosition().y);  playerShape.move(0, -playerShape.getGlobalBounds().height);  break;  }  else {  init();  }  }  }  }  if (velocityY < 0)  {  if (levelData[row][col] == 1)  {  if (playerShape.getGlobalBounds().intersects(level[row][col].getGlobalBounds()))  {  init();  }  }  }  if (levelData[row][col] == 2)  {  if (playerShape.getGlobalBounds().intersects(level[row][col].getGlobalBounds()))  {  init();  }  }  }  }  if (playerShape.getPosition().y > 600)  {  init();  }    window.clear();  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  window.draw(level[row][col]);  }  }  window.draw(playerShape);  window.display();  timeSinceLastUpdate = sf::Time::Zero;  }  }  }  };  int main()  {  Game game;  game.init();  game.run();  return 0;  } |