Lab 8 – Collision Detection

It is recommended you create a new project for this week's work; copying across the classes and files you need. This means you won't lose your previous work as we make changes to some of the base classes. Files you will need include; main.cpp, game.h/.cpp, input.h/.cpp, sprite.h/.cpp, vector.h/.cpp. Update the Sprite class with the code below. This code adds in functionality for handling collision detection.

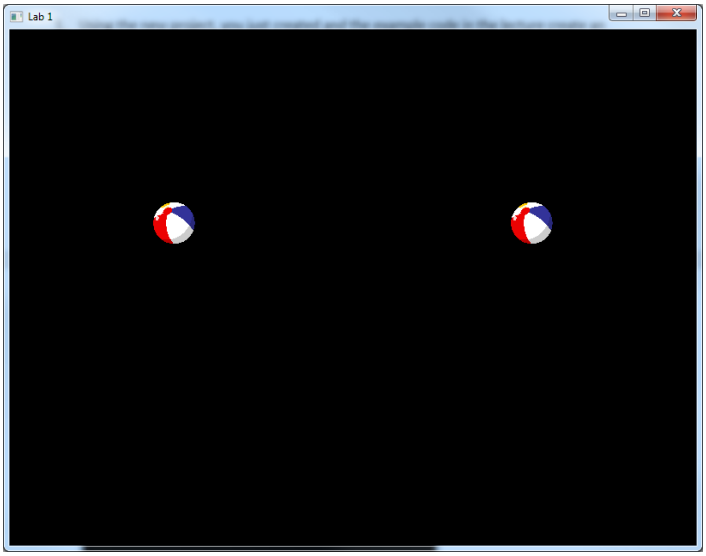
Sprite.h

```
#pragma once
#include "SFML\Graphics.hpp"

class Sprite : public sf::RectangleShape
{
public:
	Sprite(const sf::Vector2f & size = sf::Vector2f(0, 0));
	~Sprite();

	virtual void update(float dt)=0;
	void setVelocity(sf::Vector2f vel);
	void setVelocity(float vx, float vy);
	sf::Vector2f getVelocity();

	sf::FloatRect getAABB();
	virtual void updateAABB();
	virtual void collisionResponse();


protected:
	sf::Vector2f velocity;
	sf::FloatRect AABB;
};
```

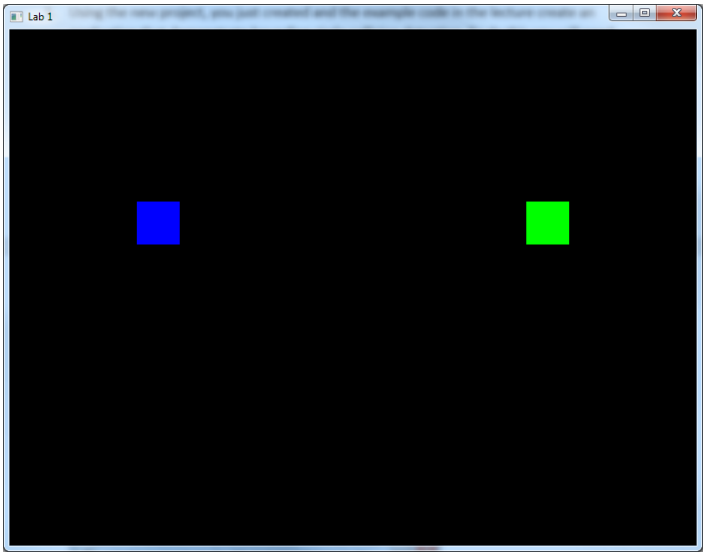
Sprite.cpp

```
#include "Sprite.h"

Sprite::Sprite(const sf::Vector2f & size) : RectangleShape(size)
{}

Sprite::~Sprite()
{}

void Sprite::setVelocity(sf::Vector2f vel)
{
	velocity = vel;
}
void Sprite::setVelocity(float vx, float vy)
{
	velocity.x = vx;
	velocity.y = vy;
}

sf::Vector2f Sprite::getVelocity()
{
	return velocity;
}

sf::FloatRect Sprite::getAABB()
{
	return AABB;
}

void Sprite::updateAABB()
{
	// Axis Aligned Bounding Box, based on sprite size and position.
	// Shape could be smaller/larger and offset if required.
	// Can be overwritten by child classes
	AABB.left = getPosition().x;
	AABB.top = getPosition().y;
	AABB.width = getSize().x;
	AABB.height = getSize().y;

}

// Reponse function, what the sprite does based on collision
// Colliding object is passed in for information
void Sprite::collisionResponse()
{
	// e.g. compare sprite positions to determine new velocity direction.
	// e.g. checking sprite type (world, enemy, bullet etc) so response is based on
that.
}
```

1. Using the new project, you just created and the example code in the lecture create an application that demonstrates bounding circle collision detection. To do this you will need two sprites that move towards each other. Add a new function to the Game class that checks if two Sprites are colliding, using the bounding circle logic (can be found in the lecture). In the game update() function check to see if the two sprite have collided, if so, have the sprite's change movement direction.

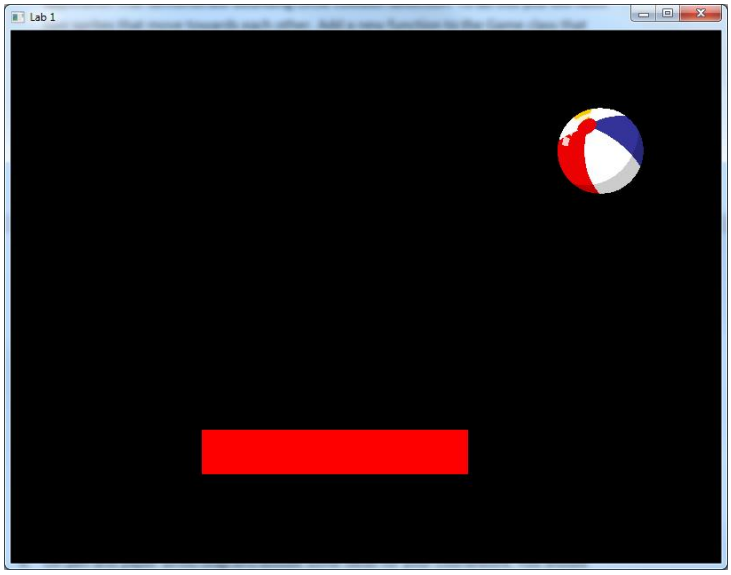


2. Using the example of AABB given in the lecture, add another function to the game class to handle AABB collision detection. Similar to the lecture example, build an application that has two sprites that move towards each other and bounce off each other, but use the AABB collision detection method rather than the bounding circle.



3. Using AABB build an application that has a player controlled paddle that can move along the bottom of the screen (similar to Breakout or Pong, see image below). The paddle should move left/right based on user interaction (left/right arrow keys). Create a Sprite that bounces around the window and bounces off the paddle. If the sprite hits the bottom of the

window it’s position should be reset to the top of the window.



4. On pen and paper write/diagram/doodle some ideas for your coursework. You should consult the coursework brief and attempt to have a coursework design that meets as many of the requirements as possible. If you have concerns your planned coursework is too/not complex enough use lab time to speak with staff about your idea.