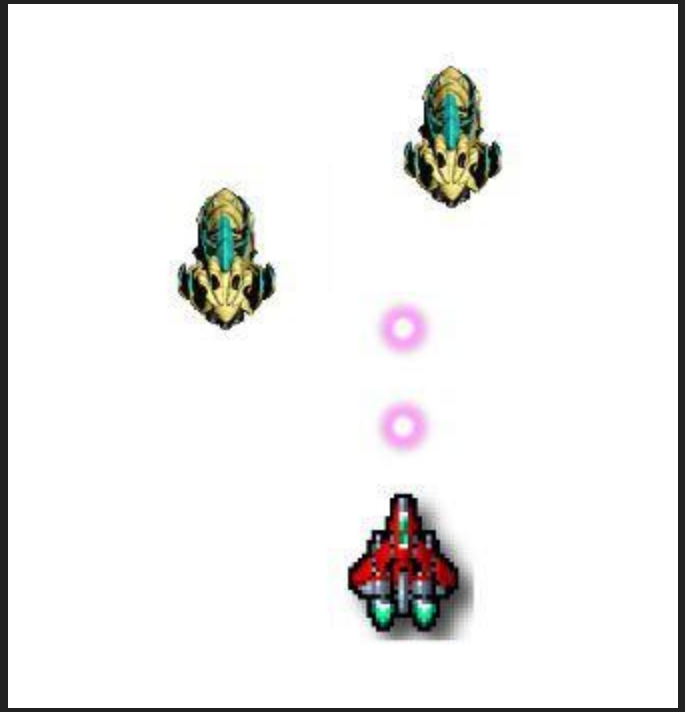
VGP230

Lesson 3 (HelloSpaceShooter)

Instructor: Darren Waine

Lesson Overview

- Enemies
  - Movement - basic “AI”
- 2D Collision
  - Circle on Circle
- Game States
- In-class Project: continuing on with HelloSpaceShooter



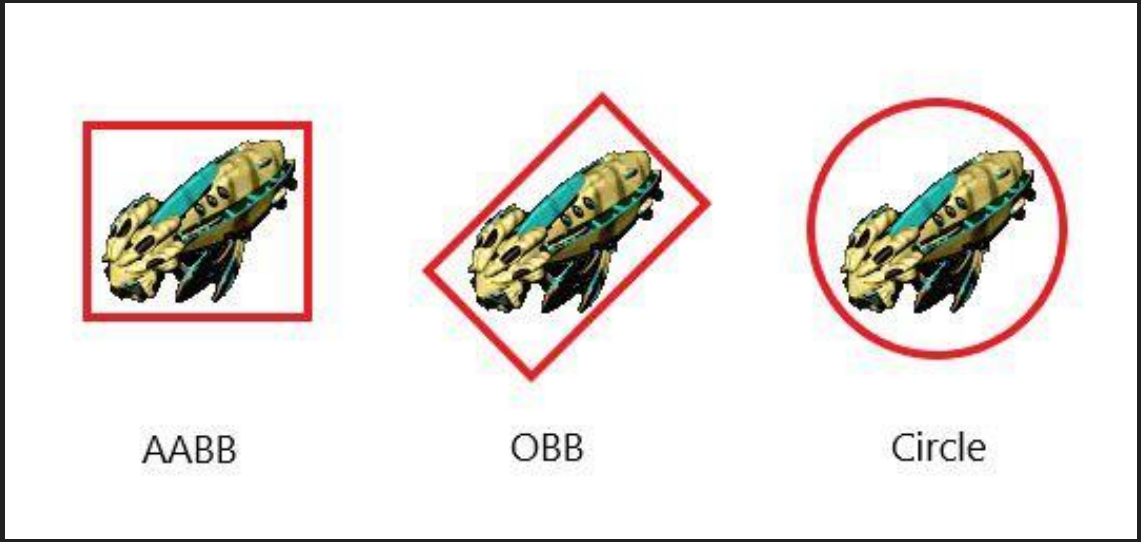
Enemies

- Enemies are a good way to add a challenge to any video game and are a common theme for 2D games specifically (Mario jumping on Goombas, Link chopping down Moblins etc).
- Enemy movement - how can we make enemies feel “alive”?
  - Randomization is a good way to make enemies seem like they are doing something on purpose, and can add challenge to the game.
  - For your midterm project - a good improvement to what we’ve done in class could be adding in a layer of AI, such as having the enemies change direction when they’ve collided with another enemy or with a wall.
- To kill an enemy, we can check to see if our bullets have collided with one.

2D Collision - Introduction

- There are various forms of two dimensional collision, so how do you know which algorithm to use for determining if the gameobjects have collided?
- It depends on the types of shapes used for the bounds of a gameobject.
- Some of these include:
  - Axis-Aligned Bounding Boxes (AABB)
  - Oriented Bounding Boxes (OBB)
  - Circles
    - This is what we'll be using for this SpaceShooter project!

2D Collision Shapes

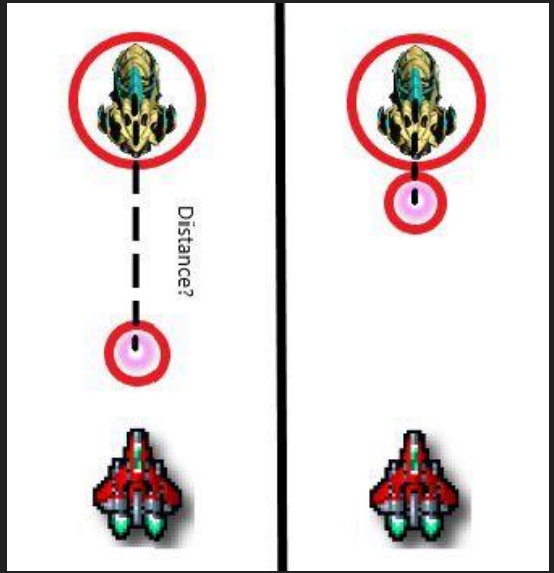


2D Collision - Circle on Circle

- The algorithm for detecting the collision between two circles is actually very simple - it's all based on distance.
- If the distance between two circle centerpoints are greater than their combined radii, they have collided!

```
distance = X::Math::Distance(bulletCircle.center, enemyCircle.center);
radii = bulletCircle.radius + enemyCircle.radius;

if (distance <= radii)
{
    // Collision!
}
```



Game States

- As we start making a more fleshed out game, we need to start thinking about game states for organizing sections of the game code.
- Each dedicated game state will be in charge of rendering and updating the game objects, UI, sound etc that fit within its category.
- Why do we want to do this? What benefit does this provide?
  - Organize the code better as it grows
  - Improve code readability
  - “Divide and conquer” in terms of fixing bugs or adding new features

Game States - cont.

- A basic set of game states might include:
  - The front end, including a title screen, maybe a “how to play section”
  - The actual gameplay state, which consists of playing the actual game
  - A lose screen, which might print the score achieved and have a way to start over