DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE

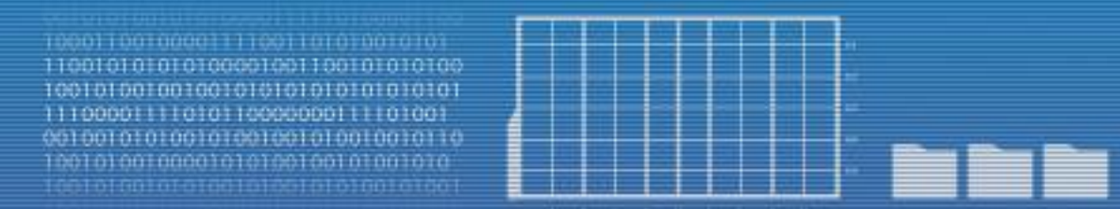
# Basic

# Applications of Physics

Aiming for Realism

# Lecture Time!

- Position: Physics-based Movement
- Friction: Motion Isn't Perpetual
- Elasticity: Boing or Splat?



DISCS

# Position

- You already know how to set an object's position

- Which also means you already know how to change an object's position
  - In response to player input
  - Automated part of program loop

# Warning

- You have been exposed to enough code that I am now assuming that I don't have to explain each and every code detail:
  - Programming conventions (ex. a variable in all caps is a constant or some pre-defined value)
  - Undeclared variables (ex. copy-pasting slide code will require you to "fill in the blanks")

# Some Adjustments

- Since we'll eventually be dealing with collisions, we need to modify our CircleShapes:

```
// assuming all CircleShapes
// have the same radius
anyCircle.setOrigin( RADIUS, RADIUS );
anyCircle.setRadius( RADIUS );
```

# Some Adjustments

- Position is already handled, but what about velocity and acceleration?
  - Each shape must have its own “current” velocity and “current” acceleration
- How do we represent velocity?
  - Hint: Similar to Position!
- How do we represent acceleration?
  - Hint: Similar to Velocity!

# Time is Money

- You will also need a global constant representing your program's time step
  - Everything has to be scaled according to this value
  - By default, this should be equal to a frame's "time" in seconds

```
#define TIMESTEP    1.0f / FPS
#define FORCE       10000.0f * TIMESTEP
```

# “Moving” an Object

- To move an object in real life:
  - Apply a force on the object
  - This causes the object to accelerate
  - Which changes the object’s velocity
  - Which changes the object’s position

# Bare Minimum

- If we didn't care about realistic physics:
  - Check for anything changing acceleration
  - Determine current acceleration
  - Add acceleration to current velocity
  - Add current velocity to position
- Simple to understand and easy to implement, but not realistic
  - It's close, though!

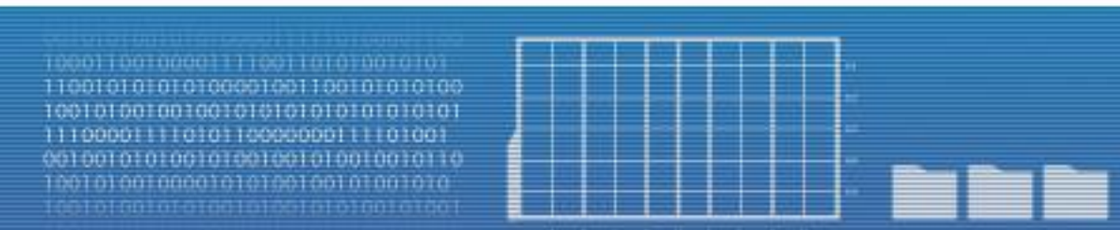
# Massive Force

- Force = Mass * Acceleration
  - How do we represent an object's mass?
  - To keep things simple, we will be using *rigid body* physics
    - Sort of like billiard balls
- Any number of forces can be acting on an object
  - They can have different directions, too

# Acceleration

- By default, an object is not accelerating
- Check for sum of all force vectors then compute for acceleration from there

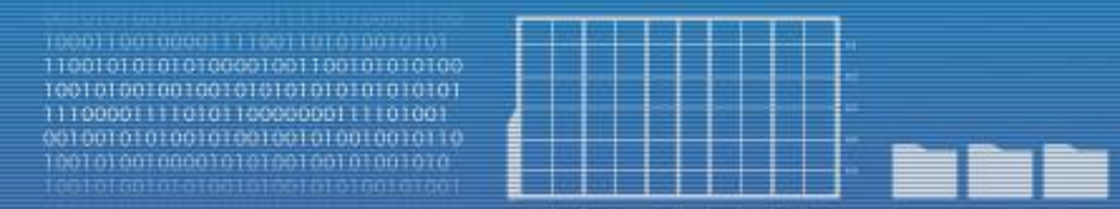
```
// acceleration initialization here - what should go here?
// process input for forces - but this is not quite correct?
if( sf::Keyboard::isKeyPressed( keyRight ) )
{       playerAccel.x += FORCE / MASS_PLAYER;
}
if( sf::Keyboard::isKeyPressed( keyUp ) )
{       playerAccel.y -= FORCE / MASS_PLAYER;
}
```



# Velocity and Position

- After that, it's a simple matter of applying physics:

```
float t = TIMESTEP;

sf::Vector2f pos = circle.getPosition();
pos = ???;              // what should go here?
circleVel = ???;        // what should go here?
```



# Velocity and Position

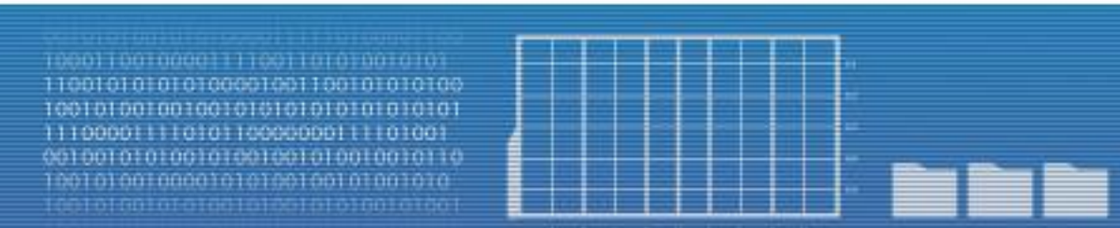
- After that, it's a simple matter of applying physics:

```
float t = TIMESTEP;

sf::Vector2f pos = circle.getPosition();
pos += circleAccel * (0.5f * t * t))
     + (circleVel * t);

circleVel += (circleAccel * t);
```

# Friction

- We can let friction cause an object to gradually lose velocity
- Friction is either on (1) or off (0)

```
// note: apply only if friction is enabled
circleVel = ???;    // what should go here?
```



# Friction

- We can let friction cause an object to gradually lose velocity
- Friction is either on (1) or off (0)

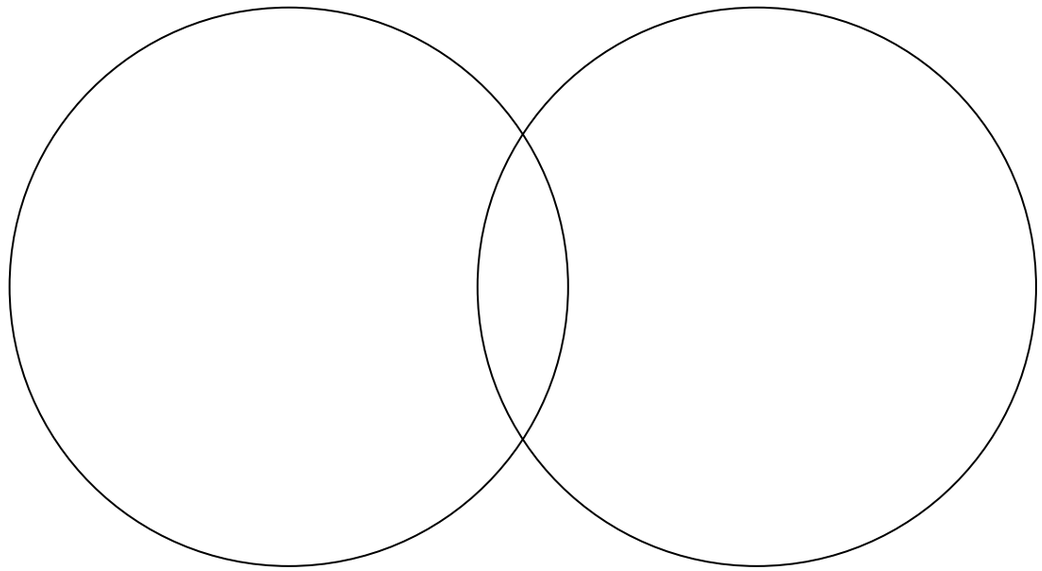
```
// note: apply only if friction is enabled
circleVel += (t * (-circleVel) /
                  MASS_CIRCLE));
```

# Elasticity

- Some objects bounce, some don't
  - Particularly important if you have some kinematic surface acting as a floor or wall
- It depends on your elasticity coefficient
  - 0 = Splat
  - 1 = Boing
  - ~~In-between = Sploing?~~

# Elasticity

- Assuming rigid body dynamics, you also have to account for penetration



- Two shapes, after movement, may be overlapping (aforementioned penetration)

# Elasticity

- This means either adjusting the object's position upon impact or only applying the collision response if the object is indeed moving towards a surface

```
// assuming an object hitting the floor
// note: this isn't treated like a force
// (more on that when we get to impulse)
circleVel.y = -ELASTICITY * (circleVel.y);
```

# Homework

- The only object active and visible should be ONE CircleShape
  - It can be any color or size
    - Should be VISIBLE
    - Should not cover the entire screen or more
    - Better to have it editable via settings.txt

# Homework

- WASD keys should apply a corresponding directional force on that CircleShape
  - CircleShape should act as though it were an air hockey puck
  - The renderable area of your program window is the air hockey arena

# Homework

- Concept of “Air Hockey”:
  - CircleShape glides along as if there is no friction (maintaining its speed indefinitely)
  - Render area of the program window are the borders/walls that the ball collides with and bounces around

# Homework

- You should be able to toggle friction on and off, though!
  - Include a visual indicator (that isn't a console printout) so you can tell if friction is on or off without moving the puck
- Suggestion: Circle color changes

# Homework

- CircleShape collision with the edges of the program window should be totally elastic
  - I may ask you to adjust elasticity
  - I may also ask you to adjust mass
  - Better to have it read from settings.txt