The Game Loop

Please make a promise…

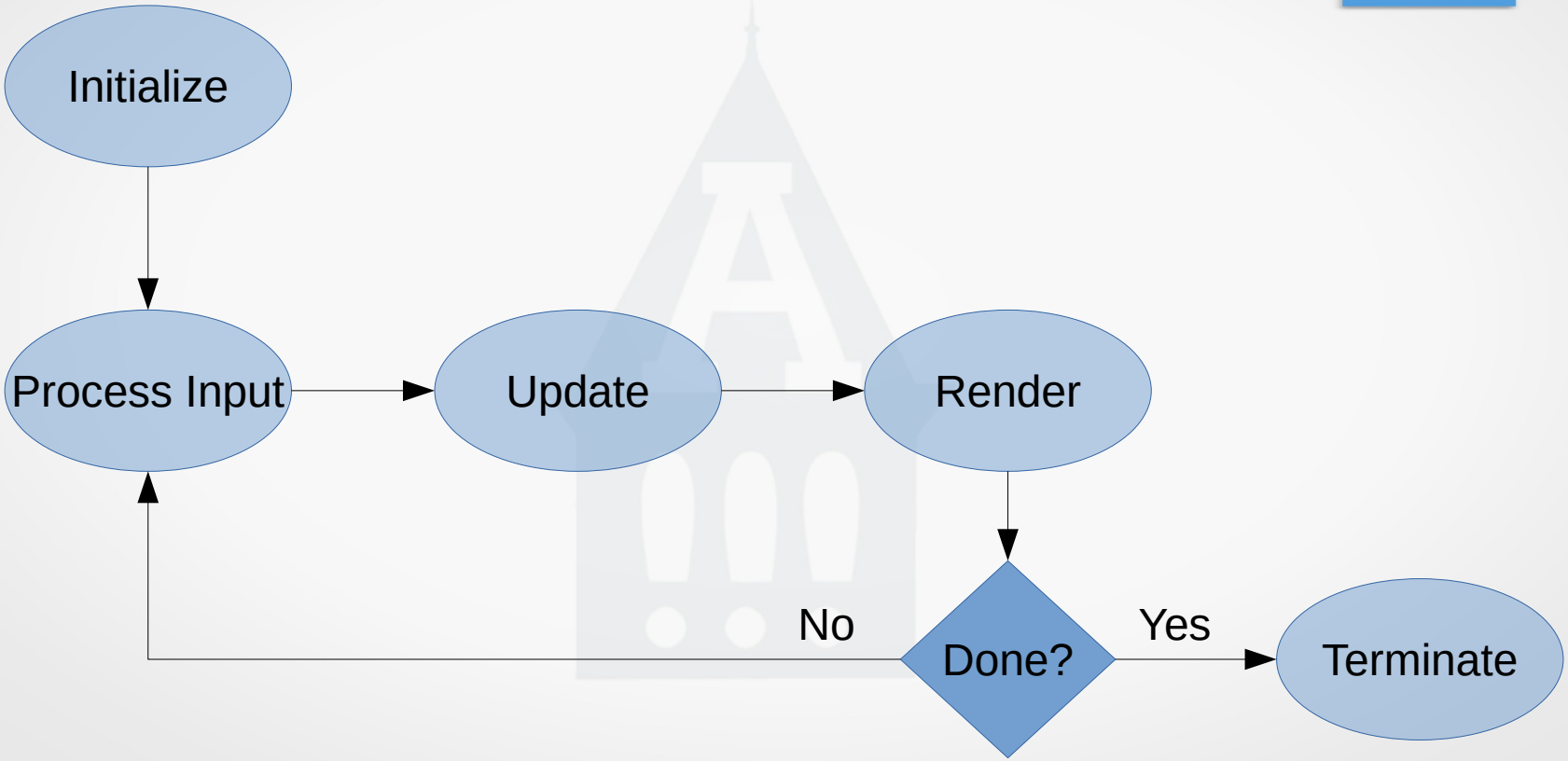
The Game Loop

Please make a promise…do not reinvent this!

The Game Loop - Step-by-Step

1. Initialize Game
   - Load graphics, models, animations, etc
   - Take initial time-stamp; call it previous time-stamp
2. Process Input
3. Update Game Logic
   - Take current time-stamp; compute elapsed time
   - Update based on elapsed time
4. Render Game State
5. Move current time-stamp to previous time-stamp
6. (if fixed rate) use spin-lock to wait until frame-time expires
7. If done, move to Step 8, otherwise return to Step 2
8. Termination

The Game Loop - Visualized



The Game Loop - MonoGame

- `Microsoft.Xna.Framework.Game`
  - `protected void Initialize()` : one time at startup
  - `protected void LoadContent()` : one time at startup
  - `protected void UnloadContent()` : one time at termination
  - `protected void Update(GameTime)` : repeatedly
  - `protected void Draw(GameTime)` : repeatedly
- You write a class that derives from `Game`, and then overrides these methods
- Note there is no `ProcessInput`, that is performed during the `Update` method
  - But ***I*** want you to write a `ProcessInput` method and call it *first* thing in `Update`
- An instance of your class is created and the `Run` method is called
  - Inside this method, a bunch of things happen, including calling the above methods

The Game Loop - MonoGame

```
protected override void Initialize()
{
     m_gameModel.initialize();
     // Set window properties here
}
```

```
public class MyGame : Game
{
     . . .
}
```

The Game Loop - MonoGame

```
protected override void Initialize()
{
     m_gameModel.initialize();
     // Set window properties here
}
```

```
protected override void LoadContent()
{
     m_gameModel.loadContent();
}
```

```
public class MyGame : Game
{
     . . .
}
```

The Game Loop - MonoGame

```
protected override void Initialize()
{
     m_gameModel.initialize();
     // Set window properties here
}
```

```
protected override void LoadContent()
{
     m_gameModel.loadContent();
}
```

```
protected override void Update(GameTime gameTime)
{
     m_gameModel.processInput(gameTime);
     m_gameModel.update(gameTime);
}
```

```
public class MyGame : Game
{
     . . .
}
```

The Game Loop - MonoGame

```
protected override void Initialize()
{
     m_gameModel.initialize();
     // Set window properties here
}
```

```
public class MyGame : Game
{
     . . .
}
```

```
protected override void LoadContent()
{
     m_gameModel.loadContent();
}
```

```
protected override void Update(GameTime gameTime)
{
     m_gameModel.processInput(gameTime);
     m_gameModel.update(gameTime);
}
```

```
protected override void Draw(GameTime gameTime)
{
     m_gameModel.draw(m_graphics, gameTime);
}
```

Timing

- Frame Rate: Measured in Hz; frames per second (fps)
- Frame Time: Amount of time within a frame Δt
  - The entire game simulation, including rendering must take place in this amount of time
  - If 60 fps, each frame has 16.66ms for everything!

Which Time?

- Wall-Clock Time: real-world elapsed time
- Simulation Time: How much game-play time has passed

- These two might be the same, but don't have to be
  - Consider Bullet-Time
    - Game frame-rate stays the same (wall-clock time)
    - Game simulation slows down (simulation time)
    - Player continues to react in real-time
      - Inputs are (probably) accepted based on wall-clock time, not simulation time)

Moving Objects

- Bad, utterly horrible, Idea: Move some number of pixels per frame
- Just as horrible Idea: Move some number of (virtual) meters per frame
- Not as bad idea, but still pretty bad: Move some number of (virtual) meters per frame based on a runntime frame-rate average
  - $X_2 = X_1 + v\ \Delta t_{ave}$
- Best Idea
  - $X_2 = X_1 + v\ \Delta t_{frame}$
- Fixed Frame Rate
  - $X_2 = X_1 + v\ 1/fps$