Moving Smartly

An Introduction to Artificial Intelligence Design in Games

Code is available here:

http://recardona.github.com/Eve

Design Tips are highlighted like this.

Rogelio E. Cardona-Rivera
Ph.D. Candidate - AI for Games, NCSU
@recardona

#ECGC2015

http://rogel.io

If you like this presentation…
(and me as a person)

Follow me on Twitter and let me know!
@recardona

Also, consider hiring me as a consultant for your next game!
http://rogel.io/consulting

About Me

- Ph.D. Candidate at NCSU
  I study AI for Procedural Content Generation of Narrative
- Puerto Rican (hablo Español)
- Likes:
  AI, Games, Narrative, Psychology, Star Wars
- Dislikes:
  Being snobby, Unnecessary Complexity, Harassment, Goobers

What this talk is about

- Discussing technical detail
  See “AI for Games” (2nd Ed.) by Millington and Funge
  for a fantastic reference/guide
- AI Design & Tradeoffs
- Demonstrating that AI is not hyper-complex
  (as some folks would have you try to believe)
- Demonstrating that AI is both science and art
- Making things that look cool

What this talk is not about

- A review of Trigonometry or Physics :(
- Optimization
- This is not the only way to do things ever™
- “Hey couldn’t you do this in a different way?”
  - The answer is yes. It depends on what you want.

Convention

- FooBar.java or FooBar have a special font because they can be found in the code we're going to look at

Overview

- Overview prototyping game ‘engine’: **Eve**
- System Architecture & Important Data Structures
- Flocking behavior
  - The `DynamicArrive` behavior & implementation
  - The `MovementBehavior` factory
  - The `DynamicBlending` behavior

The Eve Prototyping Engine

https://github.com/recardona/Eve

Checkout the “master” for the whole thing
Checkout the “demo” to follow along

- Built on top of Processing in Java (http://processing.org)
- Code-driven engine, which we will mostly skip over
- Has affordances for common game concepts (there is a setup, update, draw architecture)

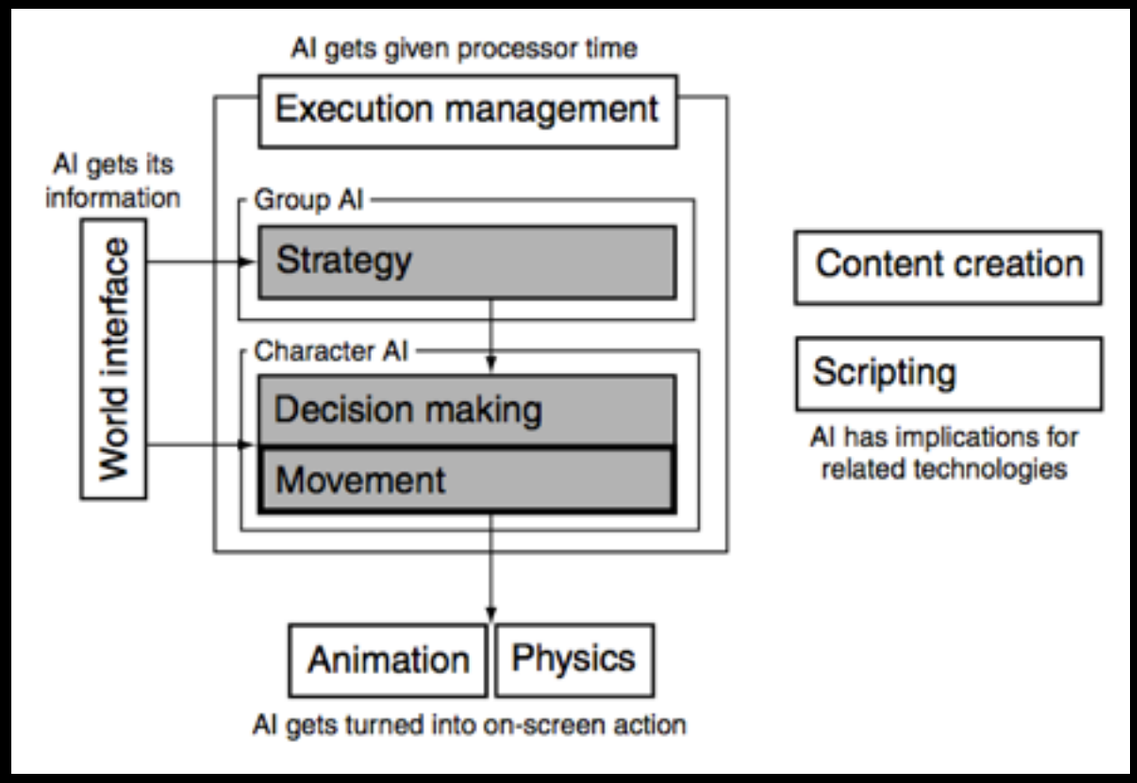
Setup and Demo Eve

https://github.com/recardona/Eve

- I use Eclipse but feel free to use whatever Java environment you'd like
- Rogelio, remember to show the final product first

Movement Behavior

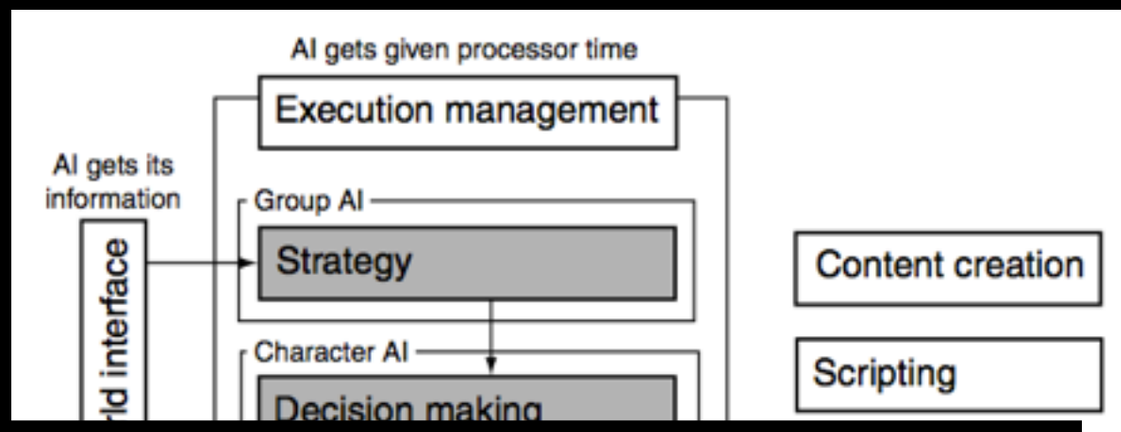
- *Movement* is the lowest level of the AI pipeline because it deals primarily with physics



- *Movement Behaviors* are similar:
  - Input: Geometric Data about the world
  - Output: **Velocities** or **Accelerations** they would like to execute

Movement Behavior

- *Movement* is the lowest level of the AI pipeline because it deals primarily with



Design Tip #1: Because all movement behaviors behave similarly, we can group them in a single interface

(go to Eclipse, Rogelio)

- Input: Geometric Data about the world
- Output: **Velocities** or **Accelerations** they would like to execute

Velocities or Accelerations?

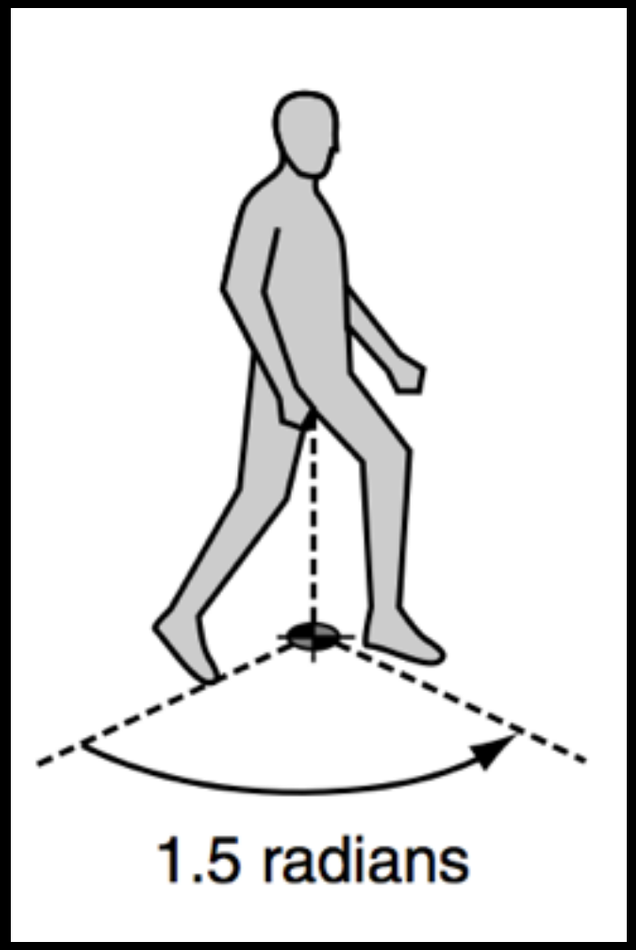
- If a MovementBehavior returns velocity information it is called Kinematic Movement
  - Typically very rigid
- If a MovementBehavior returns acceleration information it is called Dynamic Movement
  - Typically very smooth
- We will focus on Dynamic Movement

Data Structures we care about (1/2)

`DynamicSteeringOutput`

`implements SteeringOutput`

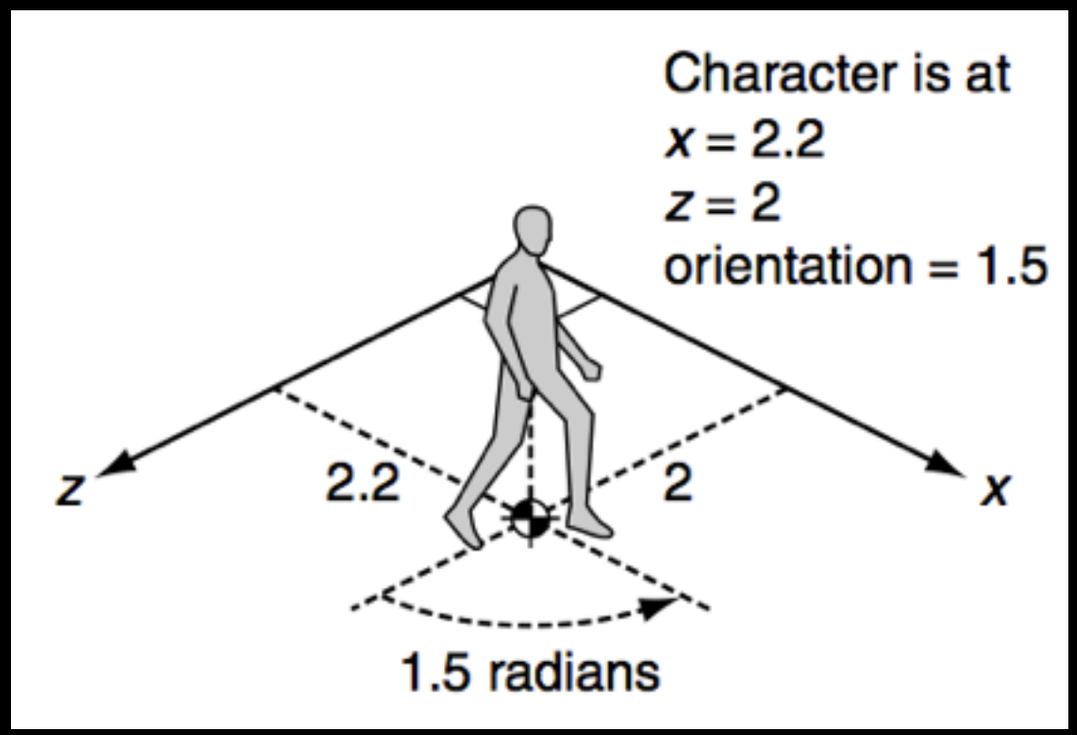
- Container for Dynamic Movement behaviors
- `linearAcceleration` (a PVector)
- `angularAcceleration` (a float value)



angular acceleration as represented by a single value

Data Structures we care about (2/2)

- `Rigidbody2D.java`
  - `position`
  - `orientation`
  - `velocity`
  - `rotation`



complex bodies represented as a single point

- AI Movement Behaviors will require information about Rigidbody2D objects.

One last thing about Movement Behaviors…

- `Rigidbody2D` scopes out the most basic ones!
  - `position matching`
  - `orientation matching`
  - `velocity matching`
  - `rotation matching`
- Movement Behaviors perform one or more of these types of calculations

One last thing about Movement Behaviors…

- `Rigidbody2D` scopes out the most basic ones!
  - `position matching`

Design Tip #2: Target the development of very basic behaviors and design ways to combine them fluidly

  - `rotation matching`

- Movement Behaviors perform one or more of these types of calculations

Flocking

Boids of a feather
flock together

- `Boid.java` 

http://en.wikipedia.org/wiki/Flocking_(behavior)

Flocking Overview

- Flocking is an emergent behavior
- At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`
- A `FlockingAgent` supports:
  1. Arriving at the Flock's center of mass
  2. Matching the Flock's velocity
  3. Separating yourself from your neighbors

Flocking Overview

- Flocking is an emergent behavior
- At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`
- A `FlockingAgent` supports:

  1. Arriving at the Flock's center of mass

  2. Matching the Flock's velocity

  3. Separating yourself from your neighbors

Dynamic Arrive Movement Behavior

Dynamic Arrive

a position-matching movement behavior

1. Get the direction to a target
2. Depending on how close we are to the target:
    2.1. Calculate a target velocity with full speed
    2.2. Calculate a target velocity with a scaled speed
    2.3. Calculate a zero target velocity
3. Accelerate to target velocity
4. Return acceleration information

How does the behavior look?

We need to look at the target!

Dynamic Look Where You Are Going

an orientation-matching movement behavior

1. Get the direction of the character's velocity

2. Calculate a target rotational velocity

3. Accelerate to target rotational velocity

4. Return acceleration information

But how do we combine them?

With a

MovementBehaviorFactory!

`MovementBehaviorFactory`

- Conceptually, it's an object that creates objects
- The ingredients are two `MovementBehaviors`
- The result is one `MovementBehavior`
  - This composite behavior is the result of combining the two `MovementBehaviors`
  - How they are combined is up to you!

`MovementBehaviorFactory`

- Conceptually, it's an object that creates objects
- The ingredients are two `MovementBehaviors`
- **Design Tip #2: Target the development of very basic behaviors and design ways to combine them fluidly**
  - This composite behavior is the result of combining the two `MovementBehaviors`
  - How they are combined is up to you!

Dynamic Arrive++

a position-matching & orientation-matching movement behavior

- Since we need to match the position of a target **and** the orientation of velocity we need an *additive combination* of movement

- We'll define a `CompositeAddBehavior!`

Much better :)

Flocking Overview

- Flocking is an emergent behavior
- At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`
- A `FlockingAgent` supports:

  1. Arriving at the Flock's center of mass
  2. **Matching the Flock's velocity**
  3. **Separating yourself from your neighbors**

Because my presentation time is finite, I will describe other behaviors conceptually.

Dynamic VelocityMatch

a velocity-matching movement behavior

1. Get the ~~direction to~~ velocity of a target
2. ~~Depending on how close we are to the target:~~
    2.1. ~~Calculate a target velocity with full speed~~
    2.2. ~~Calculate a target velocity with a scaled speed~~
    2.3. ~~Calculate a zero target velocity~~
3. Accelerate to target velocity
4. Return acceleration information

Dynamic Separate

a position-**avoiding** movement behavior

1. Identify which targets are too close
2. Calculate a repulsion strength based on how close the “too close objects” are
3. Calculate a repulsion direction
4. Accelerate in repulsion direction with repulsion strength
5. Return acceleration information

Flocking Overview

- Flocking is an emergent behavior
- At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`
- A `FlockingAgent` supports:
  1. Arriving at the Flock's center of mass
  2. Matching the Flock's velocity
  3. Separating yourself from your neighbors

Flocking Overview

- ~~Flocking is an emergent behavior~~
- ~~At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`~~
- ~~A `FlockingAgent` supports:~~
  1. ~~Arriving at the Flock's center of mass~~
  2. ~~Matching the Flock's velocity~~
  3. ~~Separating yourself from your neighbors~~

Dynamic Blending Behavior

Dynamic Blending is simple

- As you may suspect:
  DynamicBlending = weighted sum
- We need good weights for each individual behavior
  - This is where part of the art comes in

Putting it all together

Flocking Overview

- Flocking is an emergent behavior
- At a high-level, flocking emerges when each Boid follows a weighted blend of three different `MovementBehaviors`
- A `FlockingAgent` supports:
  1. Arriving at the Flock's center of mass
  2. Matching the Flock's velocity
  3. Separating yourself from your neighbors

Where does the flock go?

- Because the leader is trying to get away from the flock, and the flock is following the leader, there is a sort of tandem movement going on

et voilà

Acknowledgements

- Dr. David L. Roberts, NCSU
- The Liquid Narrative Group and its director, Dr. R. Michael Young
- The AI for Games book (2nd Ed.) by Millington and Funge

Recap!

- Important Data Structures
- Flocking behavior
  - The `DynamicArrive` behavior & implementation
  - The `MovementBehavior` factory
  - The `DynamicBlending` behavior

Recap!

- Important Data Structures
- implementation

Design Tip #1: Because all movement behaviors behave similarly, we can group them in a single interface

(go to Eclipse, Rogelio)

- The `MovementBehavior` factory

- The `DynamicBlending` behavior

Recap!

- Important Data Structures
- **Design Tip #1: Because all movement behaviors behave similarly, we can group them in a single interface**
  (go to Eclipse, Rogelio)
- implementation

Design Tip #2: Target the development of very basic behaviors and design ways to combine them fluidly

<3