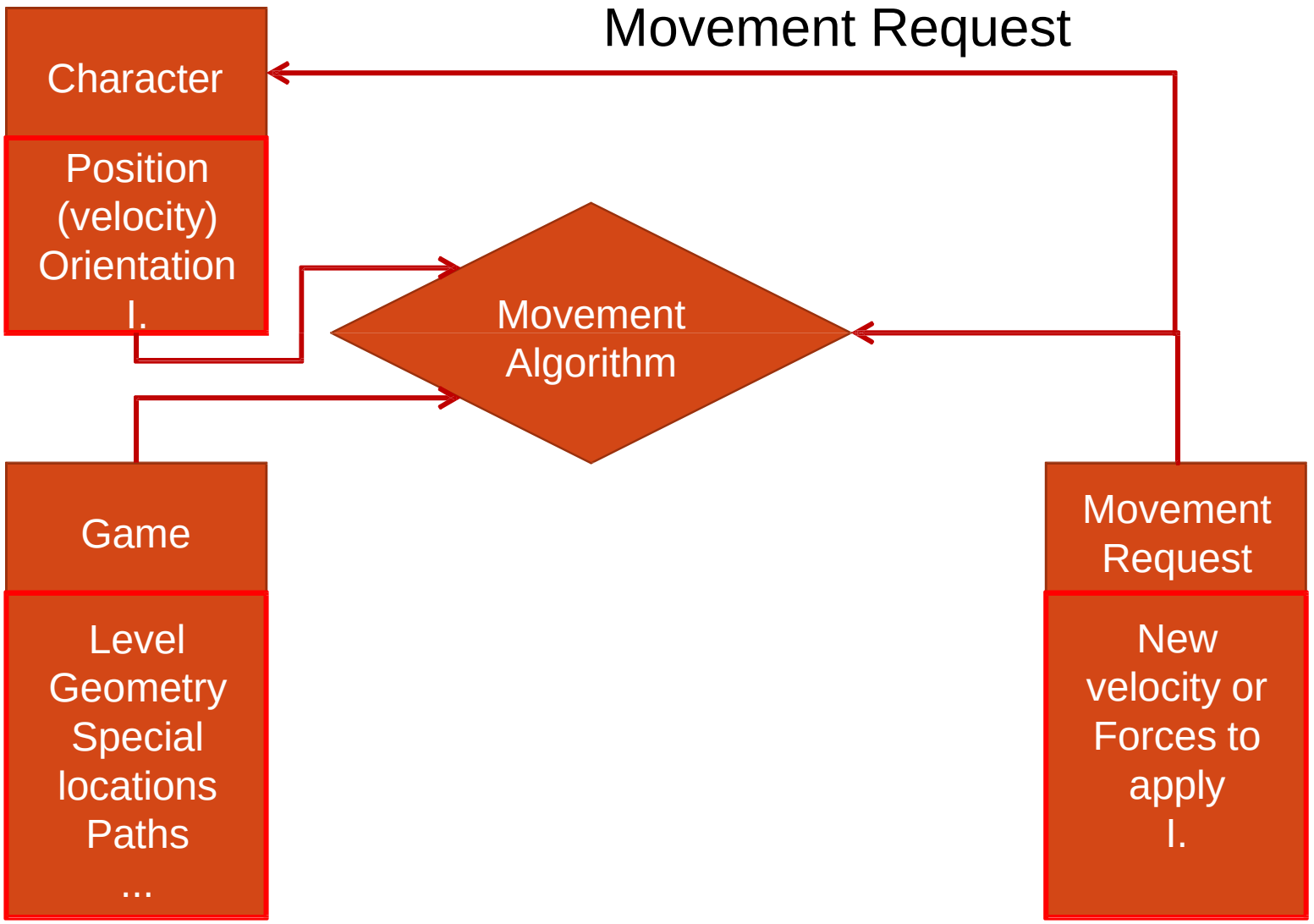
Free movement

Assumptions

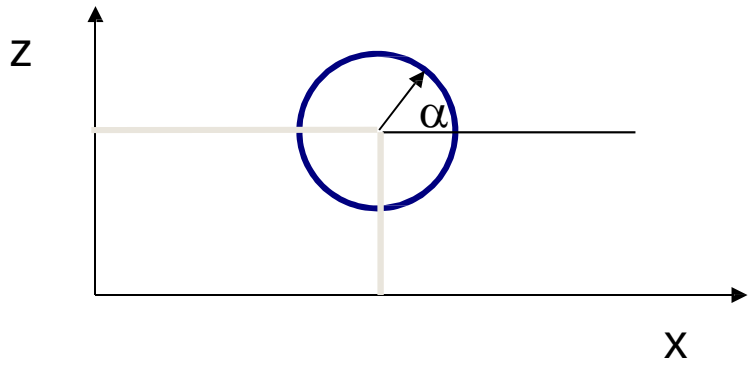
- Game character behavior needs to be computed very fast
  - Often not necessary to provide sophisticated algorithms to give impression of intelligent behavior
    - Example: Attribution of thought to Pacman's blinky character => blinky will accelere after Pac-Man eats a number of Pac-Dots.
  - Character position can be modeled as a point with orientation
  - 3Ad movement is usually not necessary
    - 2D suffices for most surface based games
    - 2½D (3Ad position and 2Ad orientation) suffices for many others

Movement Algorithm



Statics

- Character position is a point
  - Traditionally in gaming given as a vector in (x,z) coordinates
- Orientation is a 2d vector of length 1, given as ω = (sin α, cos α)



Kinematic vs. Dynamic

- Kinematic movement:
  - Adjust velocity directly using position of a character and an enemy ➔ unrealistic sometimes
- Dynamic movement (called Steering behavior)
  - Adjust by applying forces (or acceleration) on a moving object. ➔ adjusting velocity
  - Using the current velocity of a character.

Velocity를 결정

힘을 가해서 주어지는 속도
가속도를 가지고 내는 시간
어떤 프레임에는 가속도 얼마인지 결정

Kinematics 운동학

- We describe a moving character by
  - Position
    - 2Adimensional vector
  - Orientation 어디를 쳐다보고 있는지
    - 2Adimensional unit vector given by an angle, a single real value between 0 and 2 π
  - Velocity
    - 2Adimensional vector 직선 이동
  - Rotation (angular velocity) 회전각도
    - 2Adimensional unit vector given by an angle, a single real value between 0 and 2 π

Kinematics (Newton-Euler Equation)

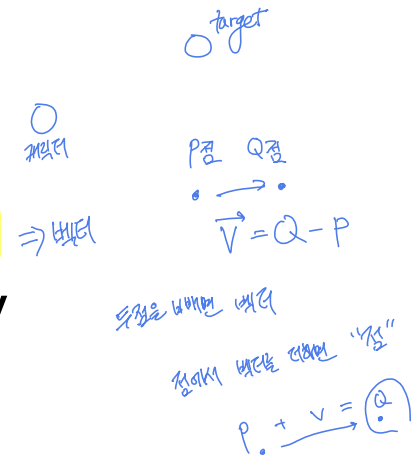
- Update calculation
  - Assume frame rate is high enough
  - Steering (i.e dynamics) is given as
    - Steering.Linear – a 2D vector
      - Represents changes in velocity
    - Steering.Angular – a real value
      - Represents changes in ~~orientation~~ Rotation
  - Update at each frame
    - Position += Velocity * Time
    - Orientation += Rotation * Time modulo (2π)
    - Velocity += Steering.Linear * Time
    - Rotation += Steering.Angular * Time

Kinematic Movement

- Uses following static data to output a desired velocity
  - Position
  - Orientation
- Can lead to abrupt changes of velocity that need to be smoothed over several frames
- Many games simplify further and force the orientation of the character to be in the direction of the velocity

Kinematic Movement

- Seek algorithm
  - Character is given a target position
  - Calculate desired direction
    - velocity = target.position – character.position
  - Normalize velocity to maximum velocity



```
def getSteering():

  # Create the structure for output
  steering = new KinematicSteeringOutput()

  # Get the direction to the target
  steering.velocity =
    target.position - character.position

  # The velocity is along this direction, at full speed
  steering.velocity.normalize()
  steering.velocity *= maxSpeed

  # Face in the direction we want to move
  character.orientation =
    getNewOrientation(character.orientation,
                      steering.velocity)

  # Output the steering
  steering.rotation = 0
  return steering
```

Kinematic Movement

- Flee: Reverse the seek velocity vector

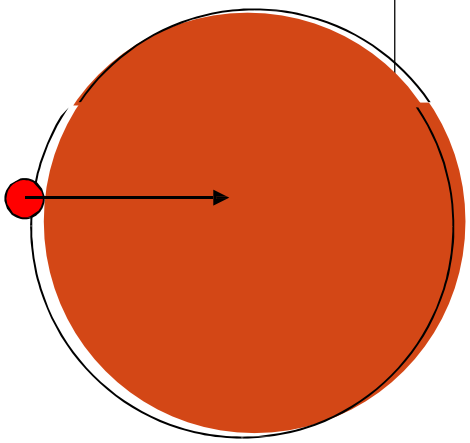
  Calculate desired direction
  - velocity = character.position– target.position
- Normalize velocity to maximum velocity

Kinematic Movement

- Seek with full velocity leads to overshooting
  - Arrival modification
    - Determine arrival target radius Lower velocity within target for arrival

```
steering.velocity = target.position –
character.position;                                    :
if( steering.velocity.length() < radius )
{
  steering.velocity /= timeToTarget;
    if(steering.velocity.length() >
MAXIMUMSPEED)
          steering.velocity *=
MAXIMUMSPEED /steering.velocity.length();
}
else { steering.velocity /=
steering.velocity.length();
        steering.velocity *= MAXIMUMSPEED;
}
                                                       )
```

Arrival Circle



Slow down if you get here

Kinematic Movement

- Wandering
  - Always at maximum speed in a forward direction (orientation에 따라 결정)
  - Direction changes
    - e.g. by small random changes

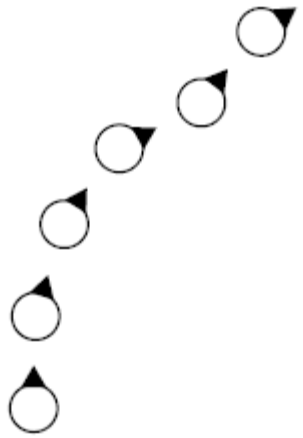
```
def getSteering():

  # Create the structure for output
  steering = new KinematicSteeringOutput()

  # Get velocity from the vector form of the orientation
  steering.velocity = maxSpeed *
    character.orientation.asVector()

  # Change our orientation randomly
  steering.rotation = randomBinomial() * maxRotation

  # Output the steering
  return steering
```



Steering Behavior

Dynamic Movement

- Steering extends kinematic movement by adding acceleration and rotational acceleration
  - Remember:

    $\mathbf{p}(t)$ – position at time t

    $\mathbf{v}(t) = \mathbf{p}'(t)$ – velocity at time t

    $\mathbf{a}(t) = \mathbf{v}'(t)$ – acceleration at time t
  - Hence:
    - $\Delta\mathbf{p} \approx \mathbf{v}$
    - $\Delta\mathbf{v} \approx \mathbf{a}$

Dynamic Movement

- Dynamic movement update
  - Acceleration in polar coordinates
    - Size of acceleration (vector length) is limited
      - In dynamic movement model, we assume that there is a strict upper bound
      - In the Newtonian model, acceleration is the result of an application of force.
    - Rotational component is also limited
      - Can be often disregarded

Dynamic Movement

- Dynamic movement update
  - Accelerate in direction of target until maximum velocity is reached
  - If target is close, lower velocity (Braking)
    - Negative acceleration is also limited
  - If target is very close, stop moving
- Dynamic movement update with Physics engine
  - Acceleration is achieved by a force
  - Vehicles etc. suffer drag, a force opposite to velocity that increases with the size of velocity
    - Limits velocity naturally

Dynamic Movement

- Position Update:

```
class Position
{
  protected: Vector2D
position;
  Vector2D velocity[2];
  double orientation,
rotation;
  friend class Steering;
  public: ...
}
```

Dynamic Movement

- Position Update:

```
class Steering
{
  private: Vector2D linear; //Acceleration   Vector
(가속도)
                  double angular; //describes changes in
orientation  (각 가속도)
  public: ...
}
```

Dynamic Movement

- Position Update:

```
INPUT: steering data structure
OUTPUT: position, orientation
void Position::update(Steering& str)
{
    this.position += this.velocity * time;
    // this.orientation += steering.rotion * time MOD 2π;
    this.orienatation = velocity.GetOrientation();

    this.velocity += steering.linear * time;
    this.rotation += steering.angular * time;
    if(this.velocity.length()) > MAXVELOCITY)
        this.velocity *= MAXVELOCITY/velocity.length();
}
```