Principles of Computer Game Design and Implementation

Lecture 26
Outline for today

• Steering behavior
  – Character model
  – Simple steering
  – Steering: combining behaviors
  – Steering in real world
A Very Rough Structure of Game AI

- Decision Making
- Planning/pathfinding
- Moving

Game engine

Today

Had a look at this
The Problem

• Decision making: *Actions* to perform
• Game engine models the world
  – One needs to link the levels

• Open space motion
  – No / simple obstacles
  – Select destination and move
    • Bound to succeed
  – *Pathfinding*

Pac-man: no pathfinding
Motion

CHARACTER MODEL
Character Position: 2D

```java
public class Model {
    Vector2f position;
    float orientation;
    ...
}
```

- Robocode
- Real-time strategies
- Platformers
Character Position: $2\frac{1}{2}$D

- Full 3D position, but
- Orientation is a single value
  - Character is upright

```java
public class Model {
    Vector3f position;
    float orientation;
    ...
}
```
True 3D

• All 6 Degrees of freedom (6DOF) are seldom used in practice
  – Complicated maths
  – Complicated controls
  – *Tilts* can be implemented in animation

• Flight simulators / space shooters
Motion

SIMPLE STEERING
Steering

- Two basic strategies
  - Seek
    - Move towards a target
  - Flee
    - Move from target

- Complex **steering**
  - In terms of basic moves
Kinematics vs Dynamics

• Recall: in computer games
  – Kinematics refers to non-realistic behaviour
  – Dynamics refers to physics-based motion
Seek: Kinematics

• Direction

\[ D = P_{\text{tar}} - P_{\text{veh}} \]

• Velocity

\[ V = D \cdot \text{normalise()} \times \text{maxSpeed} \]

• Position

\[ P_{\text{veh}} = P_{\text{veh}} + V \times \text{tpf} \]
Flee: Kinematics

• Direction

\[ D = -(P_{\text{tar}} - P_{\text{veh}}) \]

• Velocity

\[ V = D.\text{normalise()} \times \text{maxSpeed} \]

• Position

\[ P_{\text{veh}} = P_{\text{veh}} + V \times tpf \]
Seek: Dynamics

• Desired direction

\[ D = P_{\text{tar}} - P_{\text{veh}} \]

• *If differs* from current direction, apply a *steering force towards the target*

  – Use Euler steps
  – When turning
    • Consider torques
    • Align vehicle with velocity vector
Flee: Dynamics

• Desired direction

\[ D = -(P_{\text{tar}} - P_{\text{veh}}) \]

• *If differs* from current direction, apply *steering force away from* the target
Variation: Arriving

- Moving at high speed can overshoot
  - No such problem with kinematics

- When close to the target, apply brakes
Variations: Aligning and Facing

- Motion control may need to work closely with the physics engine
  - Aligning
    - Match agent’s velocity with target velocity (pursuing)
  - Facing
    - Arrive facing a direction
Complex Behaviours

- Pursue / evade
- Wander
- Separation
- Path following

Defined in terms of
- Seek / Flee
  - arriving, aligning, facing
Pursue or Intercept

• Go where target will be
  – Assume target speed does not change
    • Calculate time to get where the target currently is
      \[ t = \frac{d}{v_a} \]
    • Calculate the target position after this time passes
      \[ p = v_T t \]
  • Drive there
    – Seek(p)

Imprecise but simple
Evade

- Go away from where target will be
  - Assume target speed does not change
    - Calculate time to get where the target currently is
      \[ t = \frac{d}{v_a} \]
    - Calculate the target position after this time passes
      \[ p = v_T t \]
    - Drive from there
      - \text{Flee}(p)
Pursuing an Evading Target

Target’s speed is not constant

– Normally, cannot predict
  • Recalculate position
  • No point to use a “smarter” technique
Interpose

• Steer to midpoint of line connecting bodies
  – Bodyguard taking a bullet
  – Goalkeeper

• Similar to pursue
Opposite to Interpose

• Steer **away from** midpoint of line connecting bodies
  – Not standing in human player’s line of view
  – Not taking the lead
    • Squad behaviour

• Similar to evade
Pursue / Interpose with Offset

• Pass near but not directly into a target
  – Pursue within weapons range
  – Docking with a spaceship
  – Follow a leader in a battle formation

• Speed alignment might be necessary
Wander

1. Random steering forces
   – “wobble” around a straight line
2. *Seek* a randomly moving target

*Steering force*

Target moves randomly

More interesting behaviour
Following Paths

- **Path**: a series of *waypoints*
  - Can be open or closed (looped)
  - Locate the closest point $p_1$
  - Seek($p_1$)
  - When close to $p_1$
  - Seek($p_2$)
  - ...

Following a racetrack
Motion

STEERING: COMBINING BEHAVIOURS
Combining Steering Behaviours

• Police car:
  – Pursue
  – Avoid obstacles

• Animal
  – Wander
  – Avoid obstacles
  – Evade predators
Techniques

• Blending
  – Collect steering forces from all methods
    \[ F = w_1 F_1 + w_2 F_2 + \ldots \]
    Resulting steering force

• Priorities
  – Sort steering methods by priority
  – If higher priority method applies, use it and stop evaluation

• Hacks
Blending Example: Flocking

• A combination of:
  – Separation
  – Alignment
  – Aggregation

produces believable results

• “Batman returns” (bats & penguins) and other movies
Separation: Boid Avoidance

Move away from the boids too close
Alignment

Move in the same direction and the same velocity as the flock
Aggregation

Move towards the centre of mass of the flock
Motion

STEERING IN REAL WORLD
Collision Avoidance

• Cannot assume motion in open space

• *Steer around obstacles*
  – Cast a ray in the direction of motion
  – If collides with an obstacle
    • Apply a steering force
      – *Flee* until avoid collision

– Avoids *nearest* obstacle
– Won’t work in really complicated environments
Ray Casting

• Single ray does not notice the obstacle

• Variations:
  – Parallel side rays
  – Whiskers
  – Central ray + whiskers
Problems: Corner Trap

• Can happen with any number of rays
  – Adaptive fans
  – Special treatment of corners
Problems: Collisions with Other Movables

• Cannot avoid collision based on simple overlap test
• Collision prevention based on the intersection test is needed
Jumping

• Shooter games often use kinematics rather than dynamics for humanoids
• Jumping, however, is where this should not happen
• Tasks:
  – Locating a narrow passage to jump over
  – Selecting direction of jumping
  – Adjusting speed
Jump Points

• Level designer to decide where to jump
  – Speed alignment
  – Face
  – Seek

• Landing pads
Problems: Jump Links

• When pursuing a target, have to move in a different direction
  – Jump links
Steering Fails: Narrow Doorways

- Resulting acceleration
- Collision ray
- Route of character
- Target
Steering Fails: Long Distance

Collision ray

Route of character
Summary

• Steering is a powerful motion control mechanism

• Complex behaviours can be constructed from simple ones

• In some circumstances characters need a path to follow