Critters in Pop

CS134

Chris Pollett

Sep. 22, 2004

Introduction

- Kinds of critters
- Critter class fields
- Critter reference fields
- Critter methods
- Critter initialization

Kinds of critters

To understand why cCritter is important in Pop it is useful to look at the purpose of its various child classes (for example, in Spacewar):

- cCritterArmedPlayer: used for players in the game
- cCritterArmedRobot: used for UFOs
- cCritterBullets: used for bullets

In other games such as DamBuster, cCritterWall is used and in HW1 we used cCritterProp.

Organizing Your Projects

- For your games you will probably want to create your own critters.
- For small games, might just have one new header and one new cpp file for the whole game. Put the prototypes and class definitions in the header file and put the implementations in the CPP file.
- For larger games, have separate .h and .cpp pairs for each new critter define.

What do you want to override?

- Common methods to override are:
  - update
  - reset
  - touch
  - collide
  - die
  - damage

Critter class fields

What are the properties of critters?

- Most fields are of type int, Real, or cVector.
- Real is a typedef for a float (could be changed).
- Also, have several different kinds of references to other classes in the Pop universe.

Groupings of Critter Fields

- State fields: Real _age and int _health
- Game field: int _score and cBiota* _pownerbiota
- Position fields: cVector _position
- Velocity fields: Real _speed, cVector _velocity, and _tangent
- Acceleration, mass and force fields: Real _acceleration, Real _density, and _forcearray
- Listener field: cListener* _plistener
- Attitude and display field: cMatrix _attitude, and cSprite *_psprite

Basic Critter Fields

By thinking about what critters are for , we can come up with some of the fields they consist of.

- In order to have critters that don't stay on the screen forever, each critter has an _age field which stores how many seconds old the critter is.
- To decide whether a critter is alive or dead a _health field is used. A value of 0 indicates dead, >0 alive.
- Critters can be immortal or have a _fixedlifetime. (Ex, bullets). There is also a BOOL _usefixedlifetime to say which.

More basic fields

- To say how well a critter is doing, a _score field is used.
- To say what other critters a given critter ``see'' a cBiota pointer field *pownerbiota is given. There is a common such object for all the critters in the game.
- To do physics, critter have a _position and _velocity field and a move(dt) method that uses them. These fields are given by cVector objects which by default are 3D.

Yet more basic fields

- To be able to say where a critter can go in the world, a cRealBox _movebox field is used. This has a _locorner and a _hicorner to specify it.
- cGame has a cRealBox _border to specify size of world. By default, _movebox is the same size. To force a critter into its box could do _movebox.clamp(_position);
- If wanted the critter to be able to go off one side of its _movebox and wrap to other side could specify the int _wrapflag.

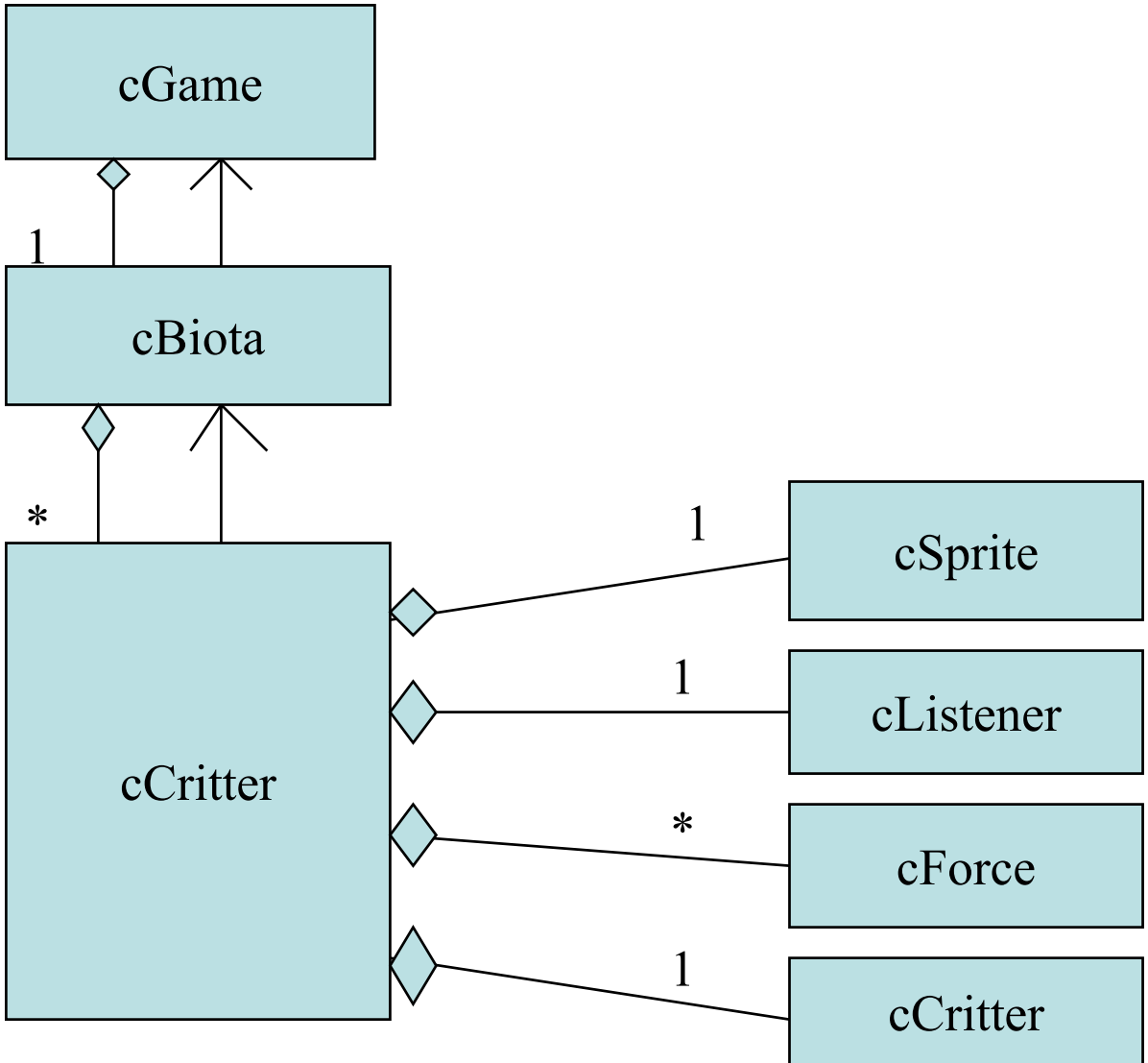
Specifying Critter Coordinate Systems

- Want to be able to give critters a maximum speed, so have Real _maxspeed. So might as well have, _velocity, _speed, and _tangent and require. _velocity = _speed*_tangent. Also, might as well store _acceleration.
- Using _tangent can create a coordinate system. Let _normal be unit vector perdicular to _tangent and in the plane given by _position, _tangent and _acceleration. The cross product _tangent x _normal gives the third coordinate directions called the _binormal.

_attitude

- This field is a four column cMatrix object that typically stores the_tangent, _normal, _binormal, and position cVector's as columns.
- can feed _attitude into the graphics pipeline so that critter always orients into the direction of motion. Ex: whales heel over when they turn.
- There is a BOOL _attitudemotionlock that turns on and off this effect.

Critter Reference Fields UML



Reference Fields Discussion

- cCritter has one owner cBiota* field called _pownerbiota. This is a back reference to a field of cGame

- cSprite _pSprite says what a critter will look like. Ex: disk-like cSpriteBubble, cPolygon, or cSpriteIcon. cSprite has a draw method which is called with _psprite->draw. Before it is called cCritter draw sends the _attitude matrix to the graphics pipeline.

More Reference Field Discussion

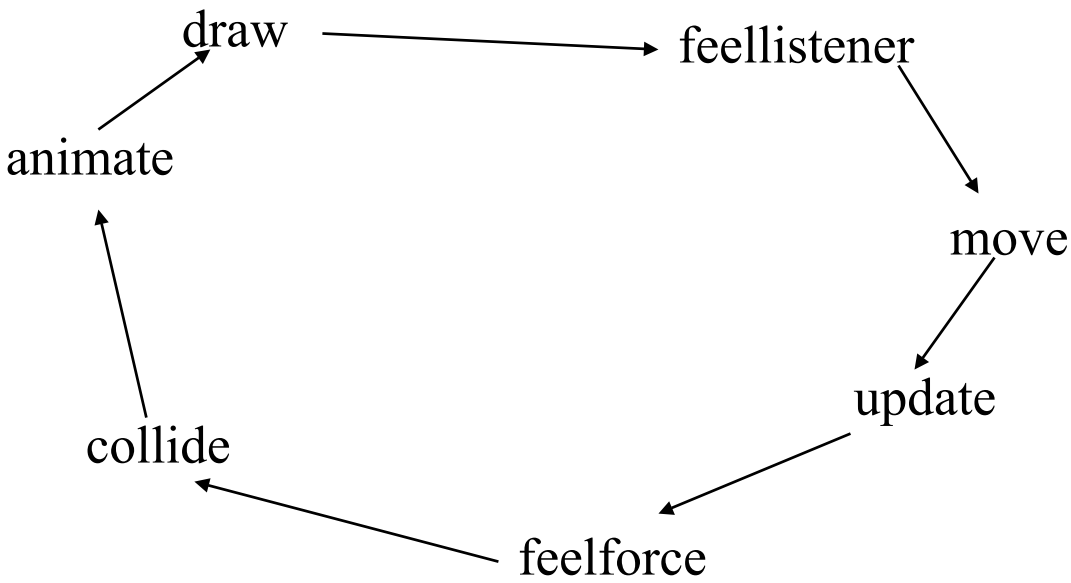
- Each cCritter has a cListener* _plistener.
- This allows critters to listen for events like mouse clicks, etc.
- The critter listens by calling its cCritter::feellistener method which in turn calls _plistener->listen.
- This is an example of the strategy pattern.

Forces and Targets

- Last day already noted critters have a CTypedPtrArray<CObArray, cForce*> field _forcearray, and have a feelforce method.
- Since to calculate acceleration we need to estimate mass, we have _mass, _density and radius field and a mass() method, radius() methods. We ensure _mass is _density *radius^3. And the radius() method just calls _psprite to figure out value.
- cCritter *_ptarget is says which critter this one is trying to get. Imposes a burden on destructor so have a virtual fixPointerRef method to solve

Critter methods

- Basic order in which cGame::step(dt) calls a cCritter's methods



update, feelforce, and feellistener

- There is a basic update(CPopView *pactive, Real dt) which determines the forces on the critter and might cause critter to die of old age. CPopView might be used to sniff the color of nearby pixels.
- Talked about feelforce last day
- feellistener(Real dt) just calls _plistener->listen(dt,this). The reference this is passed so listener can change fields of the critter. Can also get the pgame()'s cController

move

- Already know move(dt) does:
  - _velocity += dt * acceleration
  - _position += dt*velocity
- Also does:
  - Age critter by dt seconds
  - clamp the velocity to _maxspeed
  - wrap, bounce, or clamp position into the _movebox.
  - update _normal and _binormal
  - set _outcode to whichever , if any, border the critter hit: BOX_INSIDE, BOX_LOX, BOX_HIX, etc.

draw

```
void cCritter::draw(cGraphics *pgraphics, int
   drawflags)
{
    if(recentlyDamaged())
       drawflags |= CPopView::DF_WIREFRAME;
    pgraphics->pushMatrix();
    pgraphics->multMatrix(_attitude);
    _psprite->draw(pgraphics, drawflags);
    pgraphics->popMatrix();
}
```

animate

- The attitude and sprite animation are updated in this method:

```
void cCritter::animate(Real dt)
{
   updateAttitude(dt); //ck _attitudemotionlock
   _psprite->animate(dt,this);
}
```

Randomizing, die and damage

- Critter have randomizePosition and randomizeVelocity methods. There is also a mutate method.
- Each critter also has a cCritter::die method which by default just deletes the critter. Could make play a sound.
- Each critter has a cCritter:damage(int hitstrength) method which by default reduces _health by hitstrength.

collide

- Critters collide in pairs
  BOOL::cCritter::collide(cCritter *pother)
- collide is supposed to specify the reaction to a collision.
- collidesWith can be used to say which other critter this critter can collide with.

Critter initialization

- Useful to know how cCritter does initialization because when we define child classes, the base class constructor is called first.

Initialization code

```
cCritter::cCritter(cGame *pownergame):
    _pownerbiota(NULL),
    _age(0.0),
    _lasthit_age(- cCritter::SAFEWAIT),
    _oldrecentlydamages(FALSE),
    _health(cCritter::STARTHEALTH),
    _usefixedlifetime(FALSE),
    _fixedlifetime(cCritter::FIXEDLIFETIME),
    …/etc
    {
       _psprite = new cSprite();
    …}
```

Comment

Notice how the fields are initialized outside of the {…} and things that we do new for inside {…} -- this is helpful when writing the destructor so know what to destruct.