Introduction

• The class cGame
• The game’s timestep cycle
• The virtual methods of cGame
• The cBiota class
The class cGame

- In writing a game in the Pop framework you typically need to create a few subclasses of cCritter and a new child class of cGame.
- The most significant member of the cGame class is cBiota *-_pbiota. This has a collection of pointers to all the game’s active objects.
- This class is implemented as a serializable CArray, with array walking methods: draw, move, update, animate, feellistener.
void cBiota::update(CPopView *pactiveview, Real dt)
{
    for(int i=0; i < GetSize(); i++)
    {
        GetAt(i)->update(pactiveview, dt);
    }
}
More on cGame

- cGame’s also have a cCritter *pplayer for the player.
- This pointer is always assumed to be not NULL.
- pplayer is also usually a member of pbiota
- Might want to add field for other distinguished critters to the game. Ex: goals in hockey game
- A cGame also has a CRealBox _border to specify how big the world is
- The size of critters on screen depends on their radii versus border size.
- cGame’s also have a _wrapFlag saying what happens where border hit. (WRAP, BOUNCE or CLAMP)
Yet More on cGame

- Critter’s also have _wrapflags. So might think this is in instance of having same info in multiple places. (forgery)
- By default cCritter sets _wrapflag the same as its game’s _wrapflag. But subclasses might do different things.

- cGame’s also have _seedcount and _maxscore fields as well as a score() methods
- cGame’s CArray<HCURSOR, HCURSOR> _arrayCursor used to say what kind of control are available for the game
- _pcollider holds pair of critters want to be able to check if collide
The game’s timestep cycle

- step(Real dt) is probably the most important method of cGame.
- Not virtual as very delicate.
- Basically, it:
  - Adjusts game parameter (Game Over, Reset, etc)
  - Listens and passes user input to feellistener methods
  - Moves-- calls critters’ move methods
  - Updates -- calls critters’ update methods
  - Checks for collisions between pair of critters
  - Cleans up defunct critters; adds requested critters
  - Animates critter sprites
  - Draws to active views
The virtual methods of cGame

• To extend cGame you want to override the constructor and override methods like: seedCritters, initializeView, adjustGameParameters, and statusMessage
The cGame constructor

cGame::cGame():
    _seedcount(COUNTSTART),
    _gameover(TRUE),
    _maxscore(MAXSCORE),
    _scorecorrection(0),
    _wrapflag(cCritter::WRAP),
    _bDragging(FALSE),
    _pfocus(NULL),
    _pplayer(NULL),
    _border(cGame::WORLDWIDTH, cGame::WORLDHEIGHT),

...

• When override can change some of these initial values to get game you want as well as modify some of the body of constructor.
Example Modification

- For Spacewar want to change border dimension, background color and the type of player:
  \_border.set(20.0, 20.0);
  \_border.pcolorstyle()->
     >setFillColor(cColor::CN_BLACK);
  setPlayer(new cCritterArmedPlayerSpaceWar);
- You can have player which are offscreen or not member of \_pbiota: setPlayer(new cCritter(), FALSE);
- If you want to have some permanent critters can initialize in constructor. See Hw1 Solution
Seeding the Game

- cGame::seedCritters() is where initialize other critters for the game
- It is also called when game restarted or reset.
- For example, in Spacewar, the player is added in the constructor, the asteroids are added in seedCritters, and as level goes up adjustGameParameters adds UFOs.
- In Ballworld, the player and basket are added in the constructor, the ball in seedCritters.
When is seedCritters() first called?

- It is called by cPopDoc::setGameClass when the particular game in Pop is set:
  setGameClass(RUNTIME_CLASS(cGameSpaceWar));
- (Aside CRuntimeClass holds a string name of class, its size in bytes, and info about parent class. Used for serialization, run time typing)
- setGameClass:
  - constructs a new game object and puts it into _pgame field of CPopDoc
  - calls _pgame->seedcritters()
  - calls the document view to adjust their displays for the new game
    (UpdateAllViews(NULL,CPopDoc::VIEWHINT_STARTGAME, 0);
- setGameClass is the only way cGame object are constructed.
Other ways seedCritters called

• When press enter to start new game. This generates a call to cGame::reset which in turn calls seedCritters.
  – Reset also returns player health to start value and _level to 1.

• seedCritter is called within adjustGameParameters.
void cGameSpacewar::seedCritters()
{
    _pbiota-
    >purgeCritters(RUNTIME_CLASS(cCritterBullet));
    /* deleted stuff .. */
    for(int i=0; i<_seedcount; i++)
        new cCritterAsteroid(this);
}

• purgeCritters gets rid of all critters of the given type.
How the game adjusts itself

- `cGame::adjustGameParameters` gets called once per game update (which is called within `cGame::step`.)

Ex:
```
void cGameStub::adjustGameParameters()
{
    if(!health() && !_gameover)
    {
        _gameover = TRUE;
        pplayer->addscore(_scorecorrection);
        playSound("Tada");
        return
    }
    //might reseed characters etc
}
```
Initializing the view

• There are many things you might want to adjust about views: 2D versus 3D, initial viewpoint, what looking at, etc.

• Views are managed by CPopView::OnUpdate which is called from CPopView::onCreate when Pop is first launched or is called by CPopView::setGameClass
Thinking about coordinates

- x-axis goes left to right horizontally across screen
- y-axis goes bottom to top of screen
- z-axis points out of screen
- So might want to look at world from a location like: \((0.0, 0.0, 5.0)\) in a 2D game...Or might want to change where viewing world from
What OnUpdate does

if(lHint == CPopDoc::VIEWHINT_STARTGAME)
{
    pgame()->initializeView(this); //says the kind of view
    pgame()->initializeViewpoint(_pviewpointcritter);
        //says where to look within this view
    pgraphics()->installLightingModel(pgame()->plightingmodel()); /*this for now only can toggle
                lighting calculations in OpenGL, by default do calc.
                Only PickNPop doesn’t have on */
    //Call invalidate to show stuff now
}
Example view

```cpp
void cGame::initializeView(CPopView *pview)
{
    pview->setCursor(((CPopApp*)::AfxGetApp())->_hCursorArrow);
    pview->setUseBackgroundBitmap(FALSE);
    pview->setUseSolidBackground(TRUE);
    pview->setGraphicsClass(RUNTIME_CLASS(cGraphicsMFC));
    // might change in subclass
    pview->pviewpointcritter()->setTrackplayer(TRUE);
    /* Could set listener in subclass with:
       pview->pviewpointcritter()->setListener(new
cListenerViewerRide()); // works only in 3D
    */
}
```
Initializing the viewpoint critter

• Once have set the view can set up the viewpoint within this view:

```cpp
void cGame::initializeViewpoint(cCritterViewer *pviewer)
{
    if(pviewer->is3D()) // for now this check if using OpenGL
        pviewer->setViewpoint(cVector3(0.0, -1.0, 2.0), _border.center());
    else // 2D case
        pviewer->setViewpoint(cVector::ZAXIS, _border.center());
    /*
        If want to, can override and change and also set a zoom:
        pviewer->zoom(1.5)
    */
}
```
Interpreting the previous slide

cCritterViewer::setViewpoint(cVector toviewer, cVector lookatpoint)

The above call will position the viewpoint just far enough away so that every corner of the world’s _border box is visible. This method is implemented by calls to moveTo and lookAt.

If you want to see half this much change the zoom from 1 to 2.

In 2D, the toviewer is always the z-axis.
The status message

• This is the message in the status bar at the bottom of the Pop window.
• It is set by the line:
  
  cMainFrame->SetMessageText(pDoc->pgame()->statusMessage());
• The method cGame::statusMessage can be overridden by you. It returns an MFC CString object.
Example of something can put in status message

CString cStrUpdates, cStrCount;
int count = _pbiota-
          >count(RUNTIME_CLASS(cCritters));
int nUpdatesPerSecond =
int(((CPopApp*)::AfxGetApp())-
    >_timer.updatesPerSecond());
cStrUpdates.Format(“Updates/Sec: %d”,
        nUpdatesPerSecond);
cStrCount.Format(“Num critters: %d”, count);
return cStrUpdate + “ ” + cStrCount;
The randomSprite factory method

• A factory method constructs an object of a certain kind and returns pointer to it.
  An example in the cGame class is:
  cSprite* randomSprite(int spriteindex);
  /* Some allowable spriteindex’s:
    cGame::ST_SPRITETYPEPENOTUSED
    cGame::ST_SIMPLEPOLYGONS, etc */

• To use this factory method, one could within cCritter’s constructor call:
  setSprite(pownergame->randomSprite(cGame::ST_ASTEROIDPOLYGONS);

• One can override this class
The cBiota class

- cBiota is based on the class CTypedPtrArray with a few special methods added. (So Add adds a critter)
- Pointers are used to store cCritter’s so that one can use polymorphism when subclass cCritter methods.
- Arrays are a little faster than linked lists to iterate through, so that’s why arrays rather than lists used.
- cBiota’s have a _pgame pointer so that cCritter pgame() method can find the parent game.
Important methods of cBiota

- cBiota as mentioned before has a number of array walking methods: draw, move, udate, animate, render, and listen.
- Except for draw these are called by cGame::step
- draw is called by CPopView::draw
  Aside: draw traverses array backwards so that the player is drawn on top. (The player is the first element of array)
- cGame::_pfocus is used by PickNPop and DamBuilder to point to the critter being handled by the cursor. This creature is highlighted by cBiota::draw
Service Requests

• cBiota has a CArray<cServiceRequest, cServiceRequest>_servicerequestarray,
• A cServiceRequest holds a critter and a string request.
• A request might be generated when we walk through our array at some point. Don’t want to change the array size on the fly so make request.
• A typical request is to add/delete/replicate/move in array a critter.
• Deleting a critter requires a delete pcritter call and requires us to remove the invalid pointer from cBiota.
• Note ignore delete requests for player critter.