The HotCiv GUI

Instantiating the
MiniDraw Framework

[Demo]


HotCiv: Moving units ((0,0) and "stacking" are invalid)


HotCiv: Model Changes (See stdout for type)
4000 BC
Unit
Moves Left
City
Produce:
Balance:
Changing the model, mouse clicked at tile [10,7]

The Framework iteration

Learning Objectives:

- Frameworks:
  - Configure/specialize MiniDraw to support HotCiv GUI
  - See MiniDraw as example of a framework
  - See a lot of patterns in action
- TDD and Stubs
  - You can develop the GUI the TDD way
    - Keep focus & take small steps
    - No automated testing though...
  - You can develop the GUI based only on stubs

The Exercise Sequence

Basically TDD iterations

- 36.37: Observer on Game
- 36.38: Create a new **Drawing** that responds to Game state changes using the observer
  - Game → GUI integration
- 36.39: Create a **Tool** to move units
  - GUI → Game integration
- 36.40 – 36.44: Create tools to do all the other stuff
  - End of turn, change production, execute unit action, ...

i.e. *Add features* in *fast focused iterations...*

Patterns for Integration

The design

Proper design pattern protocols

- **Facade** pattern [GUI *inform* Domain]
- **Observer** pattern [Domain *inform* GUI]
- only rely on *interfaces*
  - I can develop the GUI using Stub implementations of the Domain's Facade and Observer roles
    - Facade = Interface Game
    - Observer = Interface GameListener
- reuse MiniDraw's graphical abilities
  - *let someone else do the dirty job...*

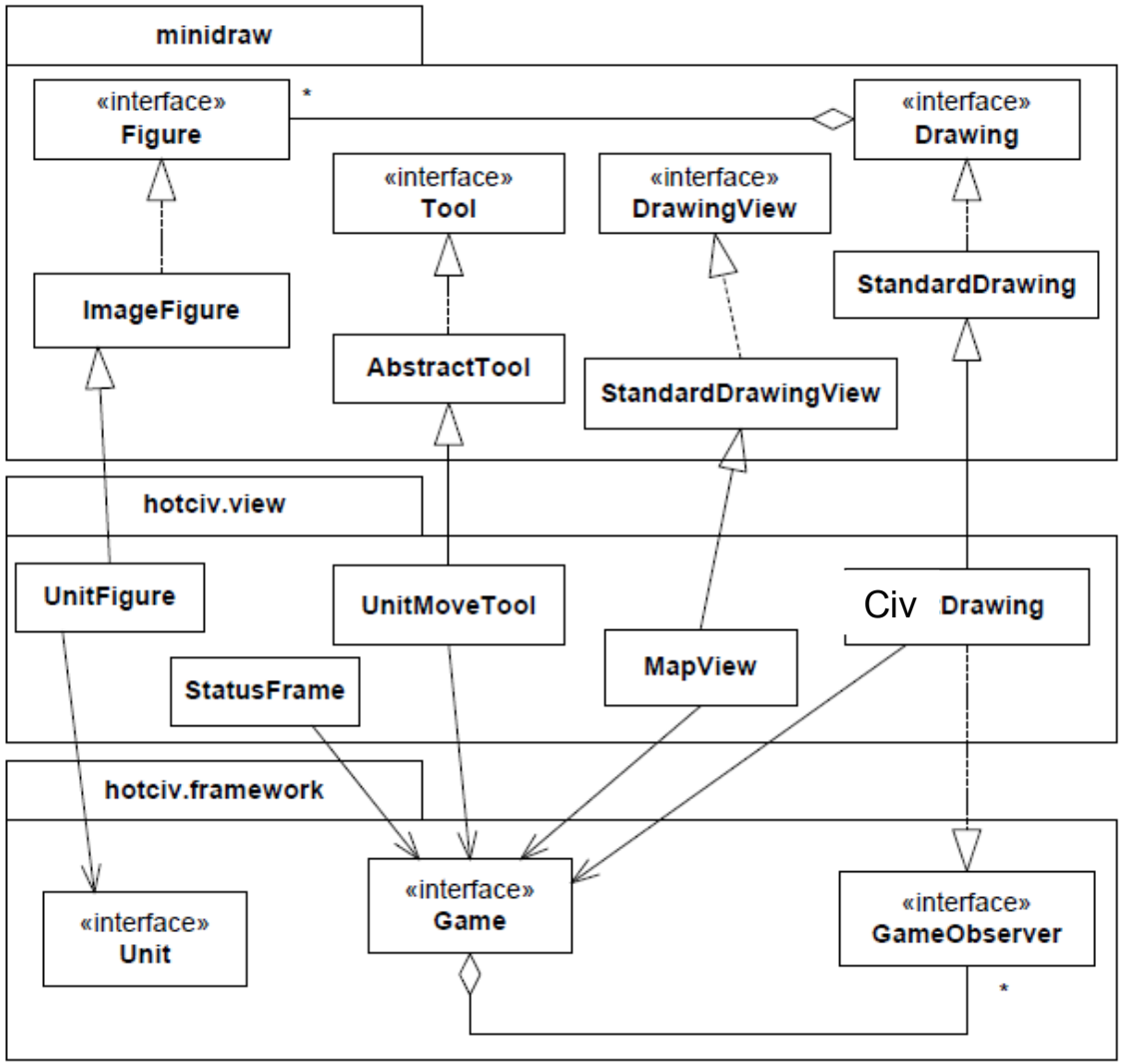
GUI

Three layers

MiniDraw

hotciv.view: HotSpots
for figures, tools, drawing and Drawingview etc

Your production code



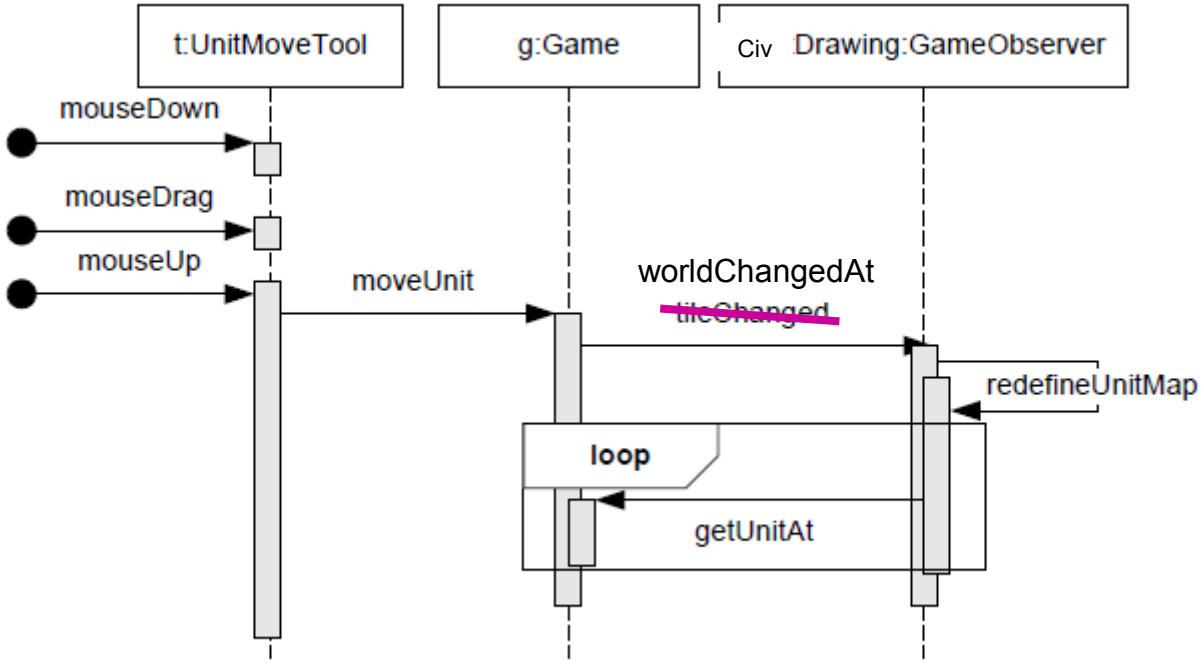
Central Protocol

The protocol between GUI and Domain:

- GUI ⇒ Domain
  - **Facade** pattern (Game interface)
  - **MVC + Adapter:**
    A tool translate mouse clicks to proper method
    - UnitMoveTool: ‘moveUnit’
    - StatusTool: ‘getCityAt’…
- GUI ⇐ Domain
  - **Observer** pattern (GameObserver interface)
  - Drawing must react upon events from domain
    - worldChangedAt()

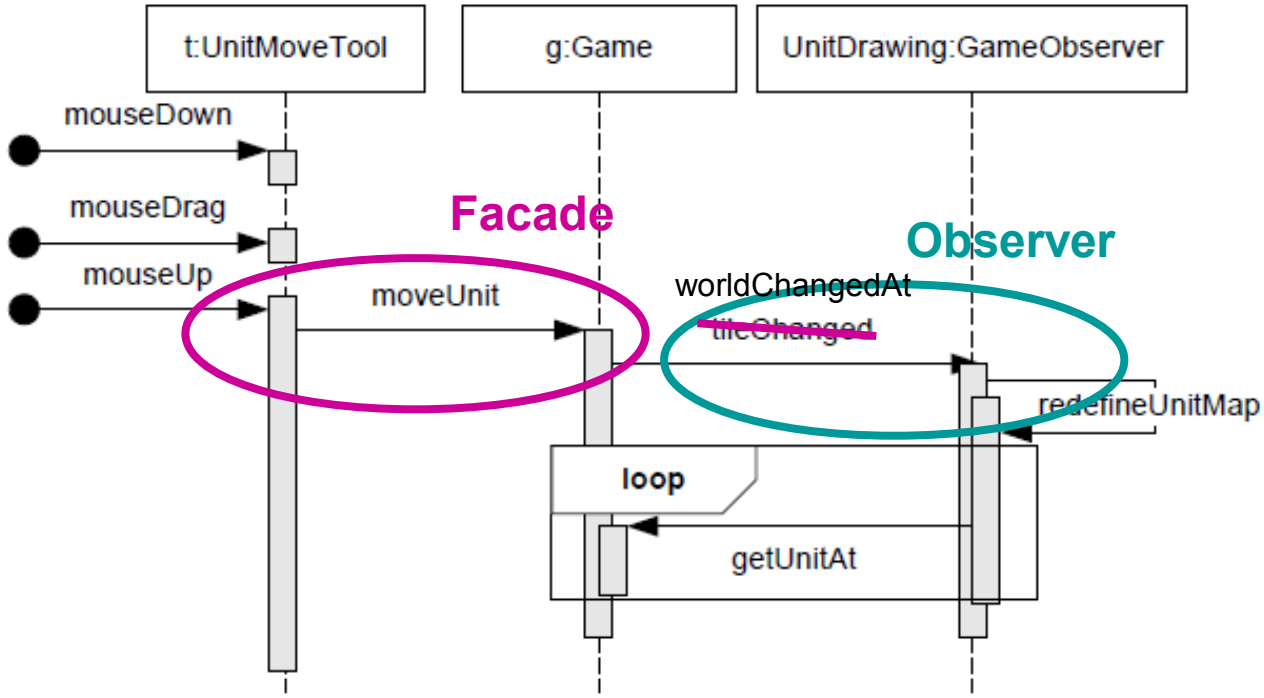
Example: Moving Units

Moving units means invoking game's "moveUnit"



The special dSoftArk provided code implements this

Example: Moving Units



GUI ⇒ Domain ⇒ GUI

Code View

```
public class CivDrawing extends StandardDrawing
  implements Drawing, GameObserver {

  /** the Game instance that this UnitDrawing is going to render un
   * from */
  protected Game game;

  public CivDrawing( DrawingEditor editor, Game game ) {
    super();
    this.game = game;

    // register this unit drawing as listener to any game state
    // changes...
    game.addObserver(this);
    // ... and build up the set of figures associated with
    // units in the game.
    defineUnitMap();
    // and the set of 'icons' in the status panel
    defineIcons();
  }
```

A compositional refactoring pending...

Brute-force redrawing ☺

```
public void worldChangedAt(Position pos) {
  System.out.println( "UnitDrawing: world changes at "+pos);
  clearSelection();
  // this is a really brute-force algorithm: destroy
  // all known units and build up the entire set again
  for ( Figure f : figureMap.values() ) {
    super.remove(f);
  }
  defineUnitMap();
}
```

Testing without production code?

The GUI can be *completely* developed without *any* real domain production code

- no Alpha, ..., Delta, nor SemiCiv

Why?

- Because I *program to an interface!*
- *Game g = ?*
- *g.moveUnit(...);*
- *g.endOfTurn();*

Example: 36.38

Goal: Implement CivDrawing

- Responsibility:
- update graphics upon game state changes

A testing tool to trigger game state changes

```
class UpdateTool extends NullTool {
  private Game game;
  private DrawingEditor editor;
  public UpdateTool(DrawingEditor editor, Game game) {
    this.editor = editor;
    this.game = game;
  }
  private int count = 0;
  public void mouseDown(MouseEvent e, int x, int y) {
    switch(count) {
    case 0: {
      editor.showStatus( "State change: Moving archer to (1,1)" );
      game.moveUnit( new Position(2,0), new Position(1,1) );
      break;
    }
    case 1: {
      editor.showStatus( "State change: Moving archer to (2,2)" );
      game.moveUnit( new Position(1,1), new Position(2,2) );
      break;
    }
    case 2: {
      editor.showStatus( "State change: End of Turn (over to blue)" );
      game.endOfTurn();
      break;
    }
    case 3: {
      editor.showStatus( "State change: End of Turn (over to red)" );
      game.endOfTurn();
      break;
    }
    default: {
      editor.showStatus("No more changes in my list...");
    }
    }
    count ++;
  }
}
```

Demo: ant update

Test Stubs

Game implementation?

- Complete façade, has not sub-objects
- Units: only three
- moveUnit: only moves red archer…

Conclusion: **simple code**

But

- *Sufficient for our sprint goal: to develop the CivDrawing that responds to unit moves, etc.*

```
public class StubGame2 implements Game {

  // === Unit handling ===
  private Position pos_archer_red;
  private Position pos_legion_blue;
  private Position pos_settler_red;

  private Unit red_archer;

  public Unit getUnitAt(Position p) {
    if ( p.equals(pos_archer_red) ) {
      return red_archer;
    }
    if ( p.equals(pos_settler_red) ) {
      return new StubUnit( GameConstants.SETTLER, Player.RED );
    }
    if ( p.equals(pos_legion_blue) ) {
      return new StubUnit( GameConstants.LEGION, Player.BLUE );
    }
    return null;
  }
  // Stub only allows moving red archer
  public boolean moveUnit( Position from, Position to ) {
    System.out.println( "-- StubGame2 / moveUnit called: "+from+"->"+to )
    if ( from.equals(pos_archer_red) ) {
      pos_archer_red = to;
    }
    // notify our observer(s) about the changes on the tiles
    gameObserver.worldChangedAt(from);
    gameObserver.worldChangedAt(to);
    return true;
  }
```

Test Stubs

Requirement:

- Enough to test all GUI related behaviour
- Example
  - Must be able to GUI's behaviour for illegal moves – therefore some moves must be invalid!

```
/** move units - all moves are valid, except
 * 1) moving to (0,0)
 * 2) moving to the location of another unit
 */
@Override
public boolean moveUnit(Position from, Position to) {
  // find out if it is a battle
  System.out.println( "Moving from "+ from + " to "+ to);

  if ( to.equals( new Position(0,0) ) ) { return false; }
```

What about TDD

Can I develop the GUI by TDD?

- **Yes**: Define a test first, like
  - "MapView": **OneStepTest:** *draw the world…*
  - "TestModelUpdate": *see game state changes reflected*
  - ...
  - and *then* implement the proper MiniDraw role implementations that makes this happen!

- **No**: I cannot make it automated as I have to visually inspect the result to verify behaviour
  - but manual tests are better than no tests!!!
  - especially when I refactored the damn thing to use MiniDraw instead of JHotDraw !!!

TDD note

The Tool abstraction comes in very handy for testing!

- Define a testing oriented tool that force some behaviour directly

Example:

- TestPartialDrawing
  - drive the implementation of the GUI responding correctly on updates from the Domain
- That is: *when I click with the mouse, make a state change and start the observer protocol update chain…*

On-the-spot Tools


```
public class TestPartialDrawing {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( "Click anywhere to see Drawing updates",
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes reflected on the GUI");

    // Try to set the selection tool instead to see
    // completely free movement of figures, including the icon

    // editor.setTool( new SelectionTool(editor) );
  }
}
```

```
class UpdateTool extends NullTool {
  private Game game;
  private DrawingEditor editor;
  public UpdateTool(DrawingEditor editor, Game game) {
    this.editor = editor;
    this.game = game;
  }
  private int count = 0;
  public void mouseDown(MouseEvent e, int x, int y) {
    switch(count) {
    case 0: {
      editor.showStatus( "State change: Moving archer to (1,1
      game.moveUnit( new Position(2,0), new Position(1,1) );
      break;
    }
    case 1: {
      editor.showStatus( "State change: Moving archer to (2,2
      game.moveUnit( new Position(1,1), new Position(2,2) );
      break;
    }
    case 2: {
      editor.showStatus( "State change: End of Turn (over to
      game.endOfTurn();
      break;
    }
    case 3: {
      editor.showStatus( "State change: End of Turn (over to
```

Test Target Demo

[Demo]: “update”

- Note archer movement
- Note also blue city
- Note status field

My Own Sprints

- Map (target ‘show’)
  - Draw a graphical world (background for figures)
  - DrawingView implementation (**MapView**)
- Unit (target ‘update’)
  - Draw units correctly positioned (Figures)
  - UnitFigure and CivDrawing
    - MiniDraw delegate all ‘model’ responsibilities to a Drawing instance. Therefore MiniDraw does not detect that CivDrawing simply brute-force erase and create lists of figures on each Game state change!
- Update (target ‘update’)
  - CivDrawing becomes **observer** on Game

Own Sprint

Move (target – is mine ☺)

- Moving graphics means moving units in Game
- UnitMoveTool, a new Tool