FOR AT KØRE HOTCIV SKAL MAN KØRE "ANT HOTCIV" TARGET

Exercise 36.36

1. Contrast HotCiv with the characteristics of frameworks (Section 32.2).

The characteristics of a framework are: Reuse of design, code, domain, can/will be reused, can be customized for user's particular need. HotCiv provides reusable code and design within a domain (here a Civilization game). Examples reside in our version of HotCiv though, where code could have been delegated to a strategy (e. g. endOfTurn()).

2. Find examples of frozen and hot spots.

Hot spots are basically anywhere in the code where a strategy is used. Examples of this would be moveUnit(), PerformUnitActionAt() and so on.

Frozen spots would be most of the methods that reside outside of the GameImpl. Examples of frozen spots inside the GameImpl would be methods like getUnitAt() and getCityAt() as these cannot be changed by framework users.

3. Find examples of inversion of control.

GameImpl calling user provided strategies.

4. Find examples of TEMPLATE METHOD in its seperation and unification variants.

There are lots of examples of template methods in its seperation form; basically any time algorithms are delegated to strategies. As far as I know there are no examples of the unification variant in our code.

5. Measure the reuse numbers (as in sidebar 32.1): how many lines of code are in your frozen spots and how many are in the hotspots, for your implemented HotCiv variants?

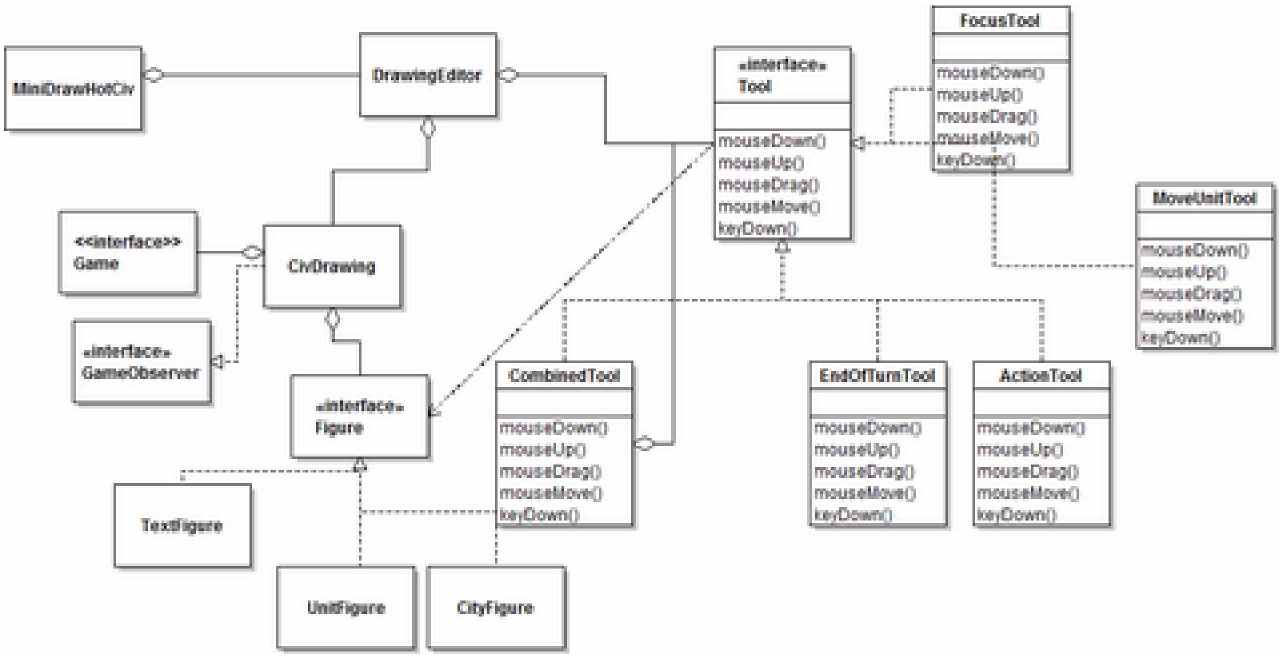
We've found ~1019 lines of variant-independent code. AlphaCiv: ~320 lines; ~70% reuse. BetaCiv: ~130 lines ~80% reuse.

Exercise 36.45

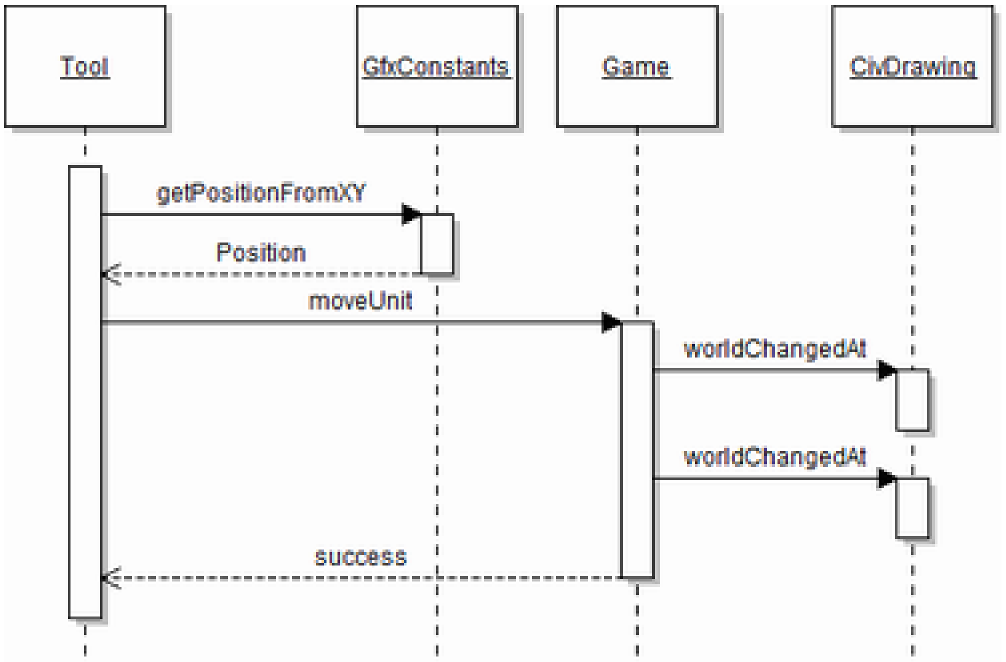
1. Outline the variability points of MiniDraw that you ahve used to make your integration, and how you have used each to design and implement a usable graphical user interface.

The variability points of MiniDraw where user-written classes have been plugged in: 5 tools, 3 figures, 1 drawing. The drawing acts as an observer on Game as well as a map of figures. This lets it auto-update the model whenever the game state changes (because GameImpl calls it to notify). I believe the figures are adapters for AWT graphics, and that they add movement functionality. We just let the drawing redraw everything on state changes however. The tools define what is to be done on mouse presses/drags/"up"'s, and using these it is possible to make calls into game when the user clicks on images of things in the game, using GfxConstants to convert x,y sets to Positions.

2. Draw a class diagram that shows your design. Emphasis should be on the classes you have developed to inject into the variability points of MiniDraw.


MiniDrawHotCiv
DrawingEditor
«interface» Tool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
FocusTool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
MoveUnitTool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
<<interface>> Game
CivDrawing
«interface» GameObserver
«interface» Figure
CombinedTool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
EndOfTurnTool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
ActionTool
mouseDown() mouseUp() mouseDrag() mouseMove() keyDown()
TextFigure
UnitFigure
CityFigure

3. draw a sequence diagram that shows the protocol between your game framework and MiniDraw when the user drags a figure of a unit to move it.


Tool
GfxConstants
Game
CivDrawing
getPositionFromXY
Position
moveUnit
worldChangedAt
worldChangedAt
success