AARHUS UNIVERSITET

Software Engineering and Architecture

HotStone GUI using MiniDraw

Finally, we may play a game!

The Framework Iteration

- *Learning Objectives:*
  - Frameworks:
    - **Use MiniDraw HotSpots to build a GUI for HotStone**
      - *Tailoring Tools (interaction), Model (domain coupling), and View (Gfx)*
    - See MiniDraw as example of a framework
    - See a lot of patterns in action
  - "TDD" as process *with visual (non-automated) testing*
    - TDD is a process:
      - Keep focus, take small steps, follow the rhythm
    - … and we will develop our GUI in that manner
    - … but have to rely on *visual/manual testing*
      - *Unfortunately, no automated testing*

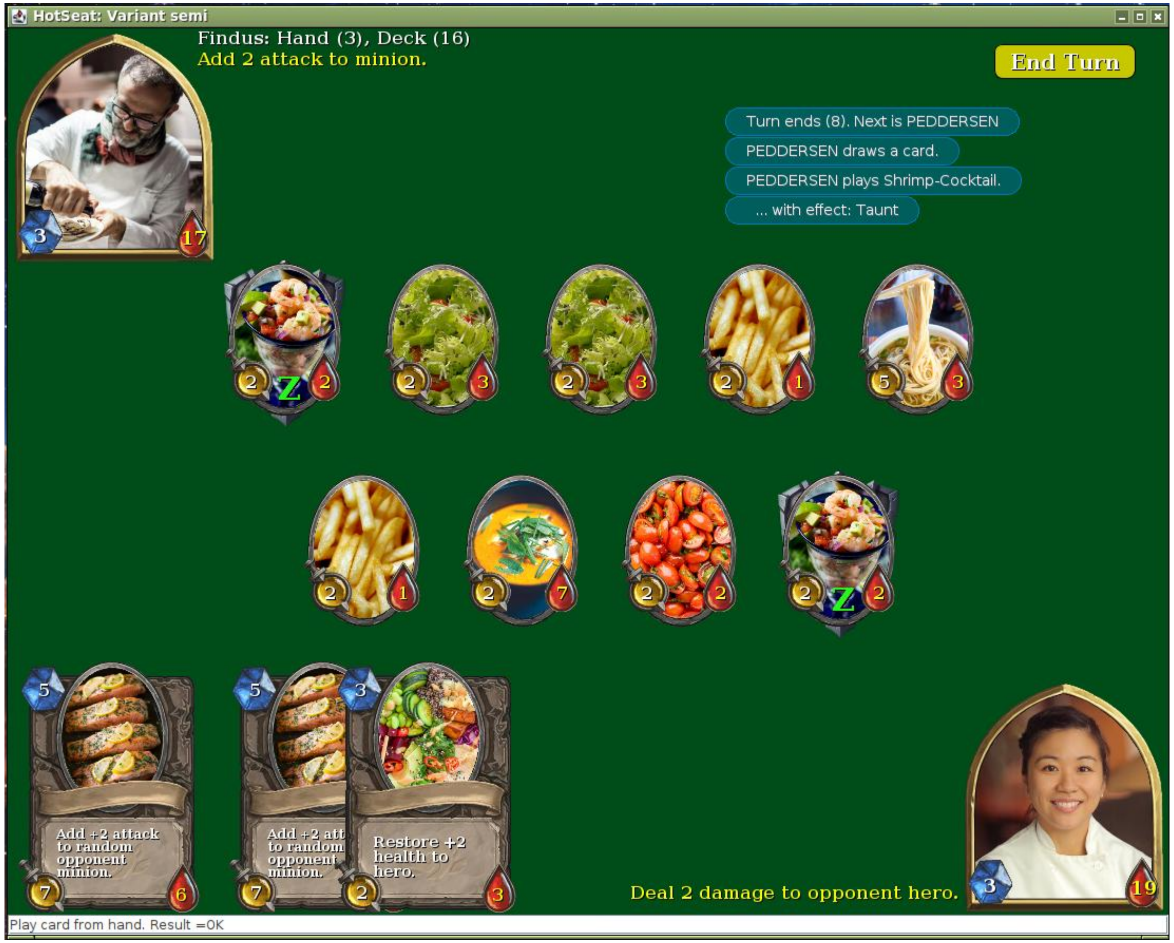
AARHUS UNIVERSITET

What will the Product be?

- A ‘hotseat’ UI for two players on a single computer

- Swap seat in front of the UI

Demo!



Findus: Hand (3), Deck (16)
Add 2 attack to minion.

Turn ends (8). Next is PEDDERSEN
PEDDERSEN draws a card.
PEDDERSEN plays Shrimp-Cocktail.
... with effect: Taunt

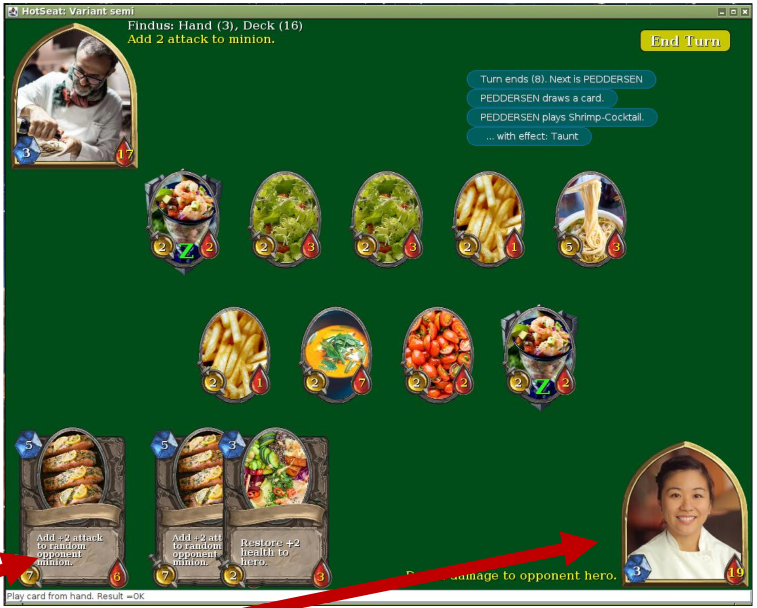
Add +2 attack to random opponent minion.
Add +2 att to random opponent minion.
Restore +2 health to hero.

Deal 2 damage to opponent hero.

Play card from hand. Result =OK

Analysis

- GUI elements are obvious, but… never-the-less:
- *Favor object composition*
  - Core = Domain = HotStone
    - Card, Hero, Player, Game
  - Edge = GUI = MiniDraw Gfx
    - Figure, Tool, Drawing, View
- **Graphical elements *representing* Game elements**
  - Card ⟺ CardFigure
  - Hero ⟺ HeroFigure
  - Etc.


HotSeat: Variant semi
End Turn
Add +2 attack to random opponent minion.
Add +2 att to random opponent minion.
Restore +2 health to hero.

Findus: Hand (3), Deck (16)
Add 2 attack to minion.

Turn ends (8). Next is PEDDERSEN
PEDDERSEN draws a card.
PEDDERSEN plays Shrimp-Cocktail.
... with effect: Taunt

D... damage to opponent hero.

Play card from hand. Result =OK

Analysis

- Thus crafting/growing a GUI …
  - (using MiniDraw or LibGdx or Unity or UnrealEngine or …)
- … entails …
  - *Drawing graphical elements* representing domain elements
  - *Translating user actions* on these Gfx elements into domain mutator calls
    - Drag CardFigure from hand up on the field ➡ playCard(…)
    - Etc
  - *Translating domain state changes* into Gfx updates
    - onHeroUpdate() ➡ HeroFigure gfx update with new health

Analysis

- In our Iteration 8 mandatory this entails…
  - Drawing Gfx
    - **MiniDraw Figure HotSpot.** **Provided by me!**

  - Translating user actions to mutator calls
    - **MiniDraw Tool HotSpot.** **Initial work provided, fill in!**
      - FRS §37.7.4
  - Translating game state changes into Gfx updates
    - **Observer on Game.** **Solved in Iteration 7**
    - **MiniDraw Drawing HotSpot.** **Initial work provided, fill in!**
      - FRS §37.7.3

AARHUS UNIVERSITET

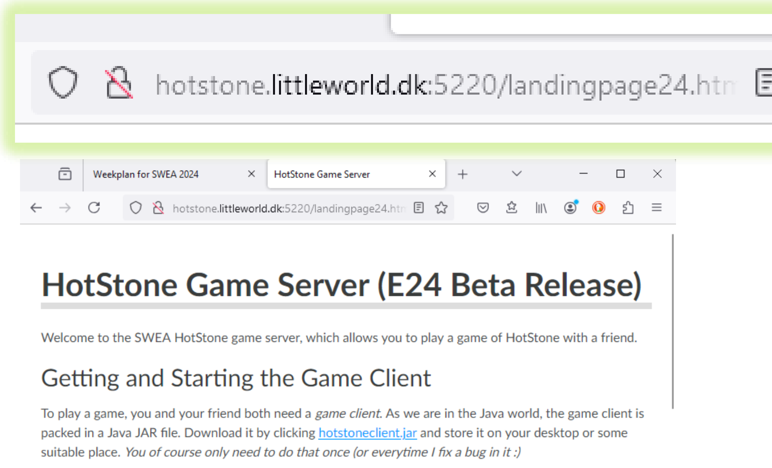
§37.7.1 User Stories

- Read FRS §37.7.1 or play the game for requirements…

1. Dragging a card from the lower left 'hand' to the field results in a call to playCard().
2. Dragging a minion from a players field onto an opponent hero or minion results in a call to attackHero() or attackCard() respectively.
3. Clicking on your own hero results in a call to usePower().
4. Clicking the upper right 'end turn' icon results in a call to endTurn().

Any state change in the game must of course also be reflected correctly in the appearance of the graphical interface:

1. Cards/minions always have the proper mana cost, attack, and health shown grapically (the latter two in the yellow attack and red blood icons).
2. Hero mana and health are shown on the hero figures in the respective icons. Power description is shown in yellow, while opponent statistics are shown in white (count of cards in hand and in deck, for the opponent player).
3. Inactive minion/hero is shown by a green Z.
4. Events/notifications from the game observer are shown in a list of blue message boxes (like the 'FINDUS draws a card.' in the figure) which are removed after a five second delay.


hotstone.littleworld.dk:5220/landingpage24.htm
Weekplan for SWEA 2024
HotStone Game Server
hotstone.littleworld.dk:5220/landingpage24.htm

HotStone Game Server (E24 Beta Release)

Welcome to the SWEA HotStone game server, which allows you to play a game of HotStone with a friend.

Getting and Starting the Game Client

To play a game, you and your friend both need a *game client*. As we are in the Java world, the game client is packed in a Java JAR file. Download it by clicking hotstoneclient.jar and store it on your desktop or some suitable place. *You of course only need to do that once (or everytime I fix a bug in it :)*

Software Architecture Views

A short detour

AARHUS UNIVERSITET

Views in Architecture

- In the software architecture field it is acknowledged that any architecture can be viewed from at least three perspectives:
  - The **runtime perspective / functional view**
    - What objects are present at runtime – how do they interact – what is the protocol?
      - Sequence diagrams and dynamics
  - The **compile-time perspective / module view**
    - What packages, interfaces, classes are there?
      - Class and package diagrams
  - The **allocation/deployment view**
    - What machines are there? What programs are running on them

Of course, highly related to each other…

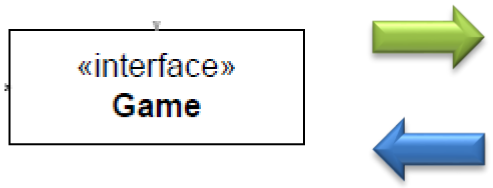
AARHUS UNIVERSITET

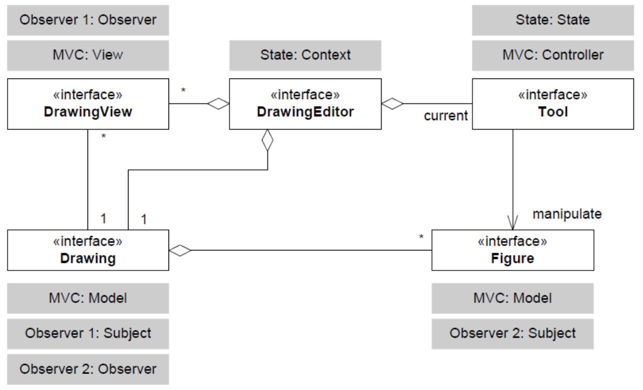
The Runtime View

Coupling Domain and GUI together

GUI ⇔ Domain

- What does this entail?

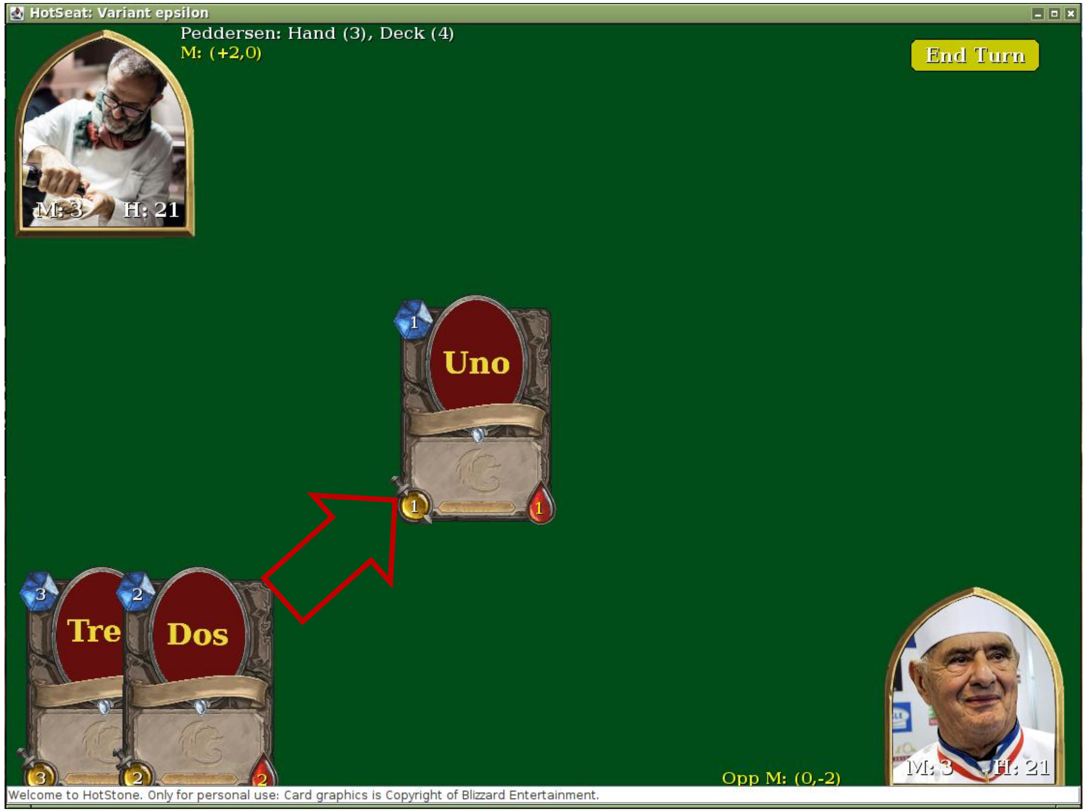




- Information flow analysis
  1. Translating *movement of Figures* into game mutations
  2. Translating *game state changes* into 2D Gfx updates
- Alas a rough *Test List* with two headlines
  - **1. From GUI to Domain**
  - **2. From Domain to GUI**

By Example (1)

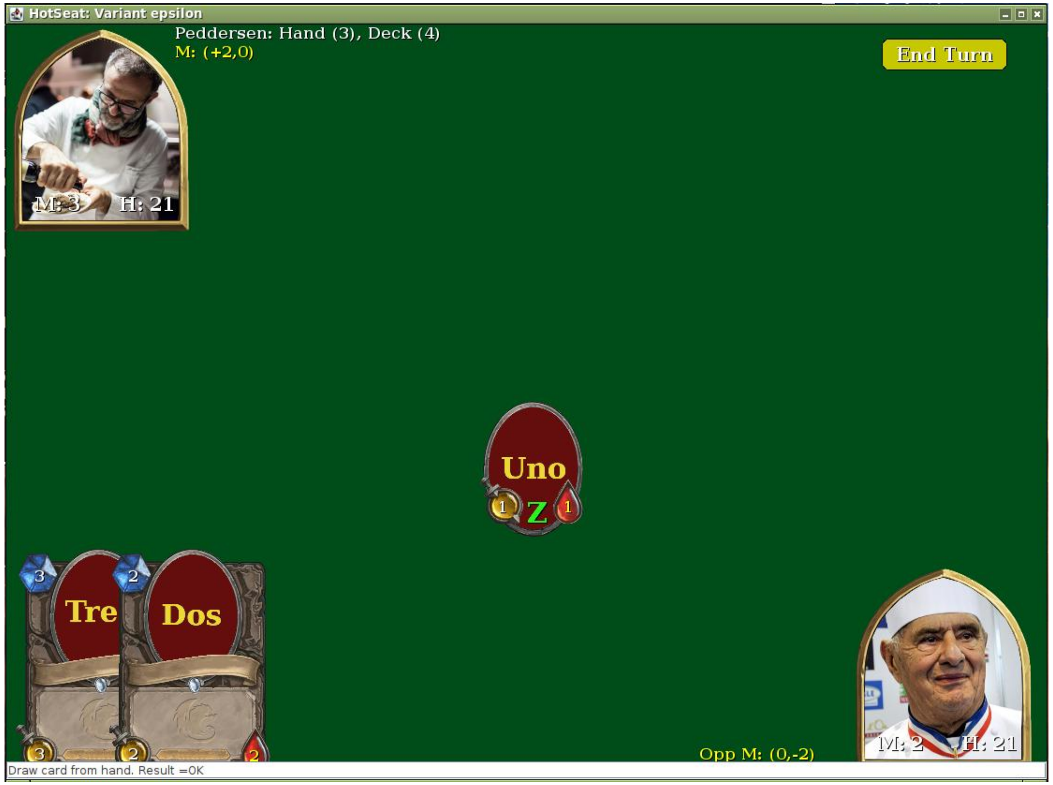
- As example:
  - *I drag 'Uno' from hand to field using the mouse*
  - Translates to
    - game.playCard(…);
- From GUI to Domain



Welcome to HotStone. Only for personal use: Card graphics is Copyright of Blizzard Entertainment.

By Example (2)

- … Which …
  - … makes Game fire a onPlayCard() event
  - … caught by MiniDraw which deletes CardFigure and adds a MinionFigure
- From Domain to GUI



Draw card from hand. Result =OK

AARHUS UNIVERSITET

Or as UML

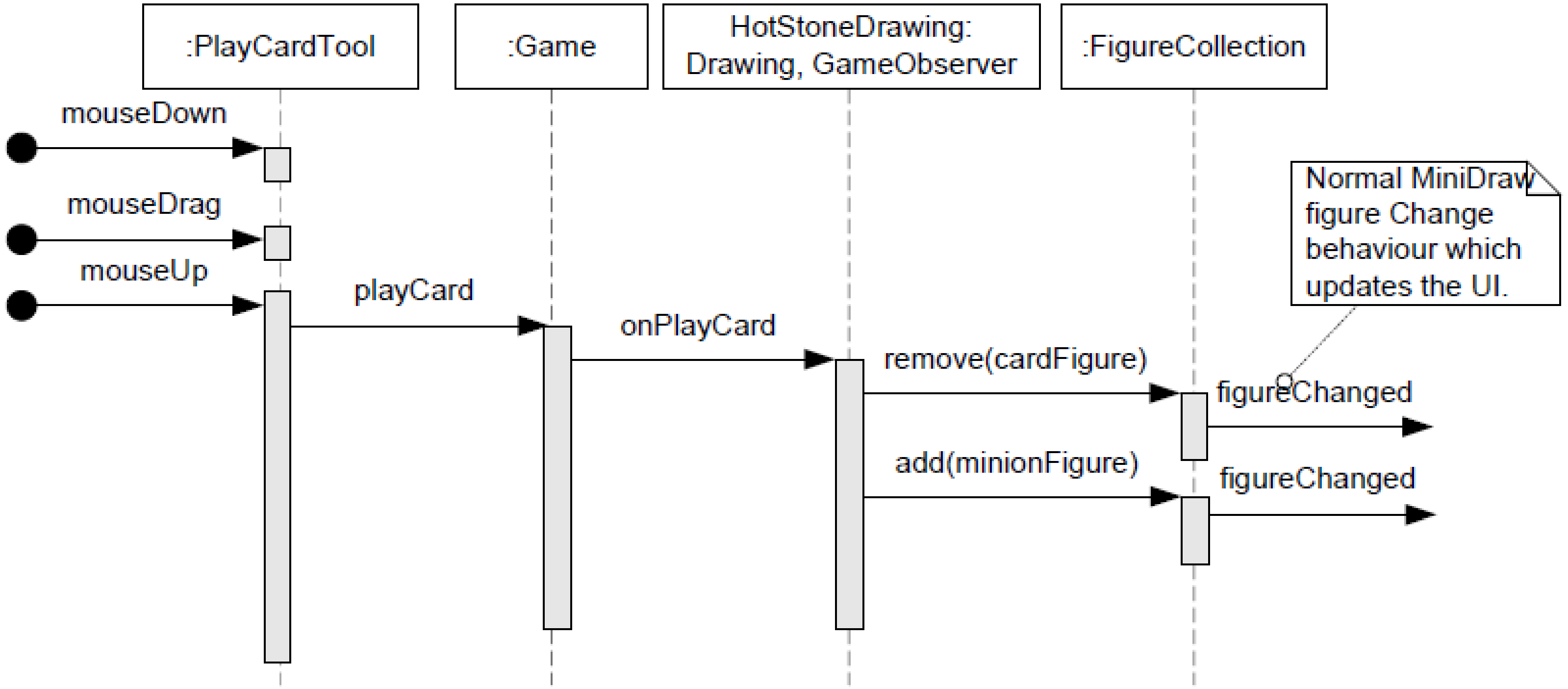
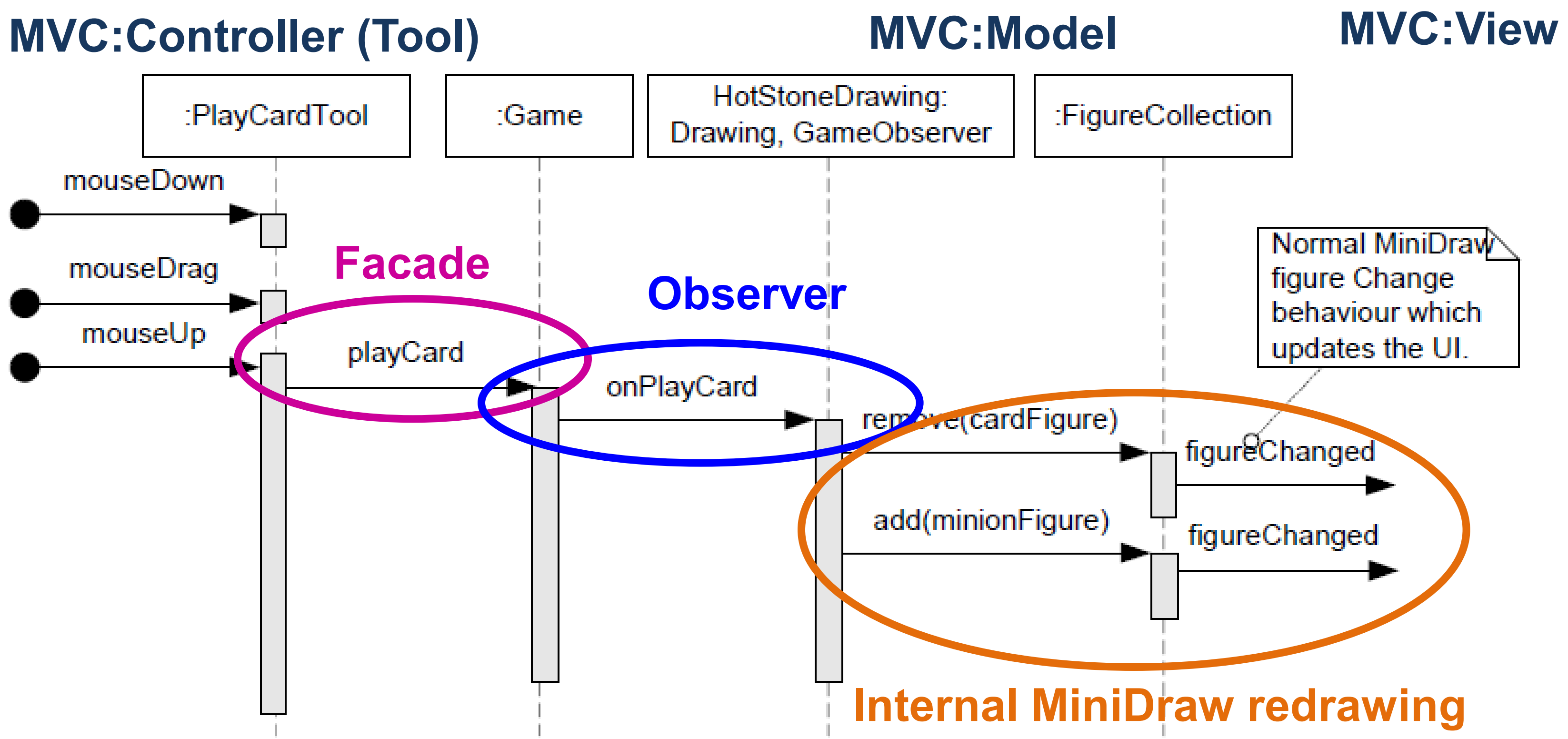


Figure 37.4: Example protocol between HotStone and MiniDraw.

Patterns Involved



GUI ⇒ Domain ⇒ GUI

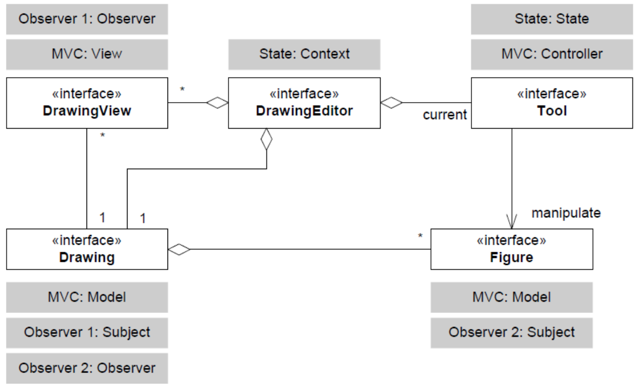
AARHUS UNIVERSITET

Divide and Conquer

- Exercise is split into two – one for each direction…

37.7.3 From Domain to GUI

«interface»
Game



37.7.4 From GUI to Domain

Exercise: Why do it in this order?

AARHUS UNIVERSITET

Domain to GUI

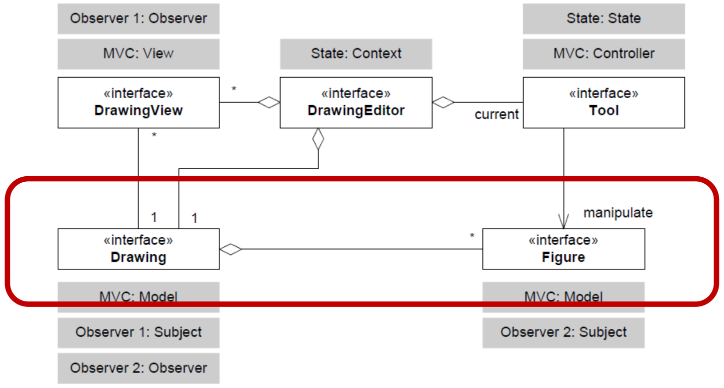
First Task

Domain to GUI

- Translating game state changes into Gfx updates
  - **Observer on Game.** **Solved in Iteration 7**
  - **MiniDraw Drawing HotSpot.** **Initial work provided, fill in!**
    - FRS §37.7.3
- What do we have?
  - The GameObserver – emitting state change events
    - Ex.: a card is drawn to hand

```
void onCardDraw(Player who, Card drawnCard);
```

- What should then happen?
  - Card appearing
    - I.e. 'f = new CardFigure(…); add(f);' in the **Drawing** role of MiniDraw



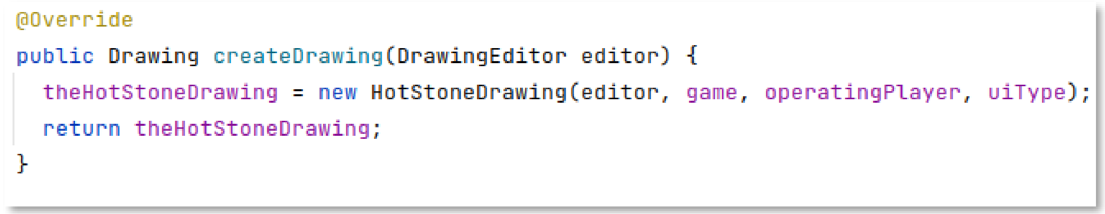
Domain to GUI

- Solution:
  - Make a special purpose **Drawing** that receives events from the Game – **Serving the Drawing role and the GameObserver role**

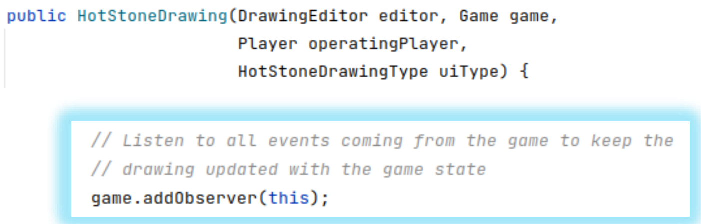
```
public class HotStoneDrawing implements Drawing, GameObserver { ... }
```

```
public void onCardDraw(Player who, Card drawnCard) {
  [create a figure with proper image for card,
    and position it correctly on the screen]
}
```

  - And A) couple that to the game and B) inject into MiniDraw



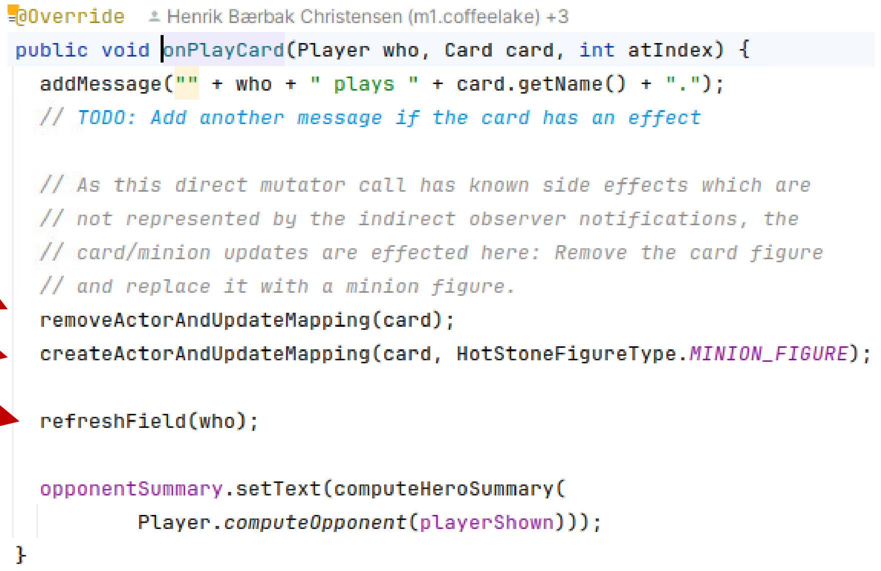
```
@Override
public Drawing createDrawing(DrawingEditor editor) {
  theHotStoneDrawing = new HotStoneDrawing(editor, game, operatingPlayer, uiType);
  return theHotStoneDrawing;
}
```



```
public HotStoneDrawing(DrawingEditor editor, Game game,
                       Player operatingPlayer,
                       HotStoneDrawingType uiType) {

    // Listen to all events coming from the game to keep the
    // drawing updated with the game state
    game.addObserver(this);
```

Domain to GUI

- Thus, reacting upon onCardDraw() is then just executing the corresponding graphical manipulations
  - onCardDraw() is part of the exercise, but another examples is:
  - onCardPlay() is implemented in provided code



```
@Override   Henrik Bærbak Christensen (m1.coffeelake) +3
public void onPlayCard(Player who, Card card, int atIndex) {
  addMessage("" + who + " plays " + card.getName() + ".");
  // TODO: Add another message if the card has an effect

  // As this direct mutator call has known side effects which are
  // not represented by the indirect observer notifications, the
  // card/minion updates are effected here: Remove the card figure
  // and replace it with a minion figure.
  removeActorAndUpdateMapping(card);
  createActorAndUpdateMapping(card, HotStoneFigureType.MINION_FIGURE);

  refreshField(who);

  opponentSummary.setText(computeHeroSummary(
          Player.computeOpponent(playerShown)));
}
```

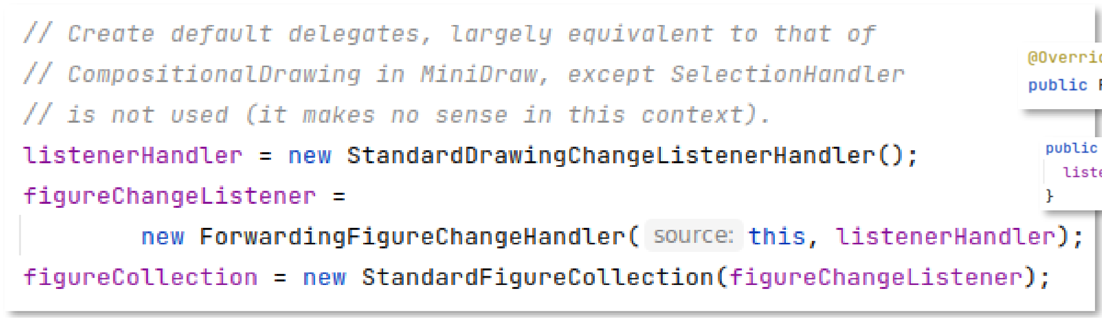
- Remove CardFigure
- Add MinionFigure
- Layout battle field

Compositional Drawing

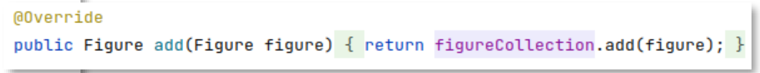
- Implementing Drawing may be tedios though

```
public class HotStoneDrawing implements Drawing, GameObserver { ... }
```

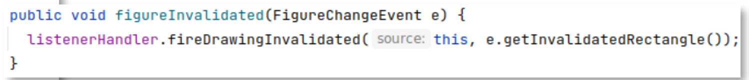
- Why – because all methods must be implemented anew
  - The collection of figures, the FigureEvent system, the …
- **Favor object composition**
  - **Reuse smaller, highly specialized, objects**
  - Thus it is done in 3 lines of code + delegation methods



```
// Create default delegates, largely equivalent to that of
// CompositionalDrawing in MiniDraw, except SelectionHandler
// is not used (it makes no sense in this context).
listenerHandler = new StandardDrawingChangeListenerHandler();
figureChangeListener =
        new ForwardingFigureChangeHandler( source: this, listenerHandler);
figureCollection = new StandardFigureCollection(figureChangeListener);
```



```
@Override
public Figure add(Figure figure) { return figureCollection.add(figure); }
```



```
public void figureInvalidated(FigureChangeEvent e) {
  listenerHandler.fireDrawingInvalidated( source: this, e.getInvalidatedRectangle());
}
```

AARHUS UNIVERSITET

GUI to Domain

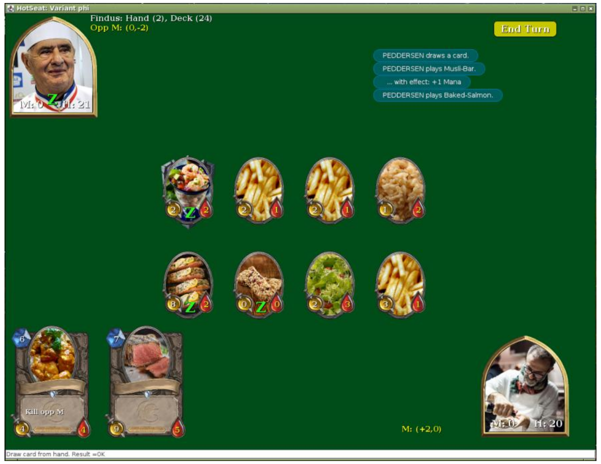
Second Task

AARHUS UNIVERSITET

GUI to Domain

- The four actions

1. Dragging a card from the lower left 'hand' to the field results in a call to playCard().
2. Dragging a minion from a players field onto an opponent hero or minion results in a call to attackHero() or attackCard() respectively.
3. Clicking on your own hero results in a call to usePower().
4. Clicking the upper right 'end turn' icon results in a call to endTurn().



PEDDERSEN draws a card.
PEDDERSEN plays Musli-Bar.
... with effect: +1 Mana
PEDDERSEN plays Baked-Salmon.

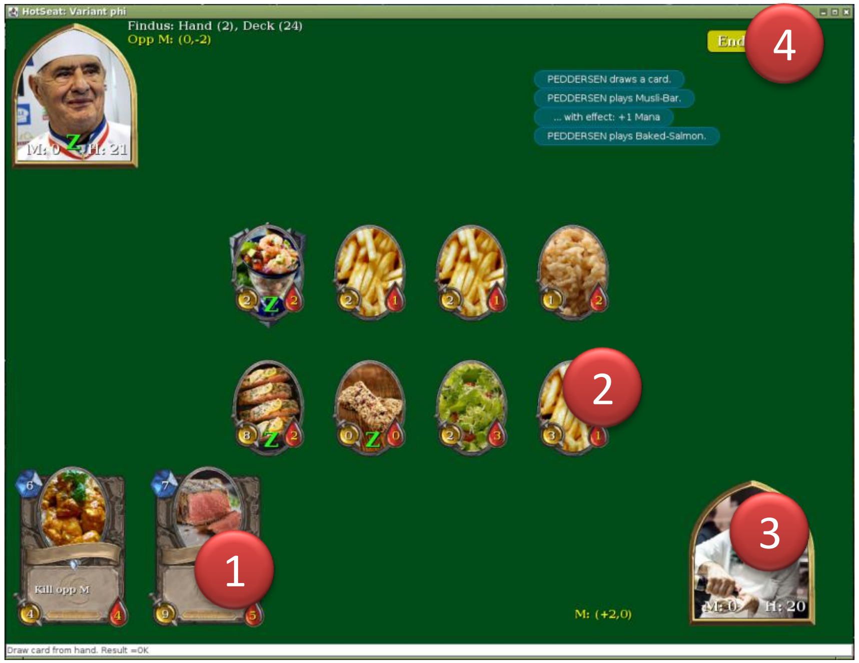
Draw card from hand. Result =OK

- Exercise
  - Given a user clicks somewhere, what determines which of the above four actions is the *wanted* action?

GUI to Domain

- Analysis
  - *Whatever the 'mouse down' event 'hits' on the GUI determines what the user wants*
    - *The (x,y) is on top of a figure*
      - *A card*
      - *A minion*
      - *A hero*
      - *A button*

1. Dragging a card from the lower left 'hand' to the field results in a call to playCard().
2. Dragging a minion from a players field onto an opponent hero or minion results in a call to attackHero() or attackCard() respectively.
3. Clicking on your own hero results in a call to usePower().
4. Clicking the upper right 'end turn' icon results in a call to endTurn().



Findus: Hand (2), Deck (24)
Opp M: (0,-2)

PEDDERSEN draws a card.
PEDDERSEN plays Musli-Bar.
... with effect: +1 Mana
PEDDERSEN plays Baked-Salmon.

M: (+2,0)

Draw card from hand. Result =OK

Selecting What To Do

- Thus
  - On mouseDown() we determine which action is relevant
  - All further mouse events must be processed accordingly
    - mouseUp() does different things

- *Change behavior according to which state system is in*
  - *Ring a bell?*

- *Exercise: What Design Pattern is Involved?*

A HotSeatStateTool

- Favor Composition, avoid The Blob
  - A state tool that delegates to sub tools

```java
public class HotSeatStateTool extends NullTool {
  [...]
  private final Tool theNullTool = new NullTool();
  private Tool state;

  @Override
  public void mouseDown(MouseEvent e, int x, int y) {
    // Find the figure below mouse (x,y)
    Figure figureAtPosition = model.findFigure(e.getX(), e.getY());
    // Iff that figure is associated with our HotStone
    // (All MiniDraw figures that handle HotStone graphics are
    // implementing the HotStoneFigure role interface).
    if (figureAtPosition instanceof HotStoneFigure) {
      HotStoneFigure hsf = (HotStoneFigure) figureAtPosition;
      // TODO: Complete this state selection
      if (hsf.getType() == HotStoneFigureType.CARD_FIGURE) {
        state = new PlayCardTool(editor, game, game.getPlayerInTurn());
      } else if (hsf.getType() == HotStoneFigureType.TURN_BUTTON ||
              hsf.getType() == HotStoneFigureType.SWAP_BUTTON) {
        state = new EndTurnTool(editor, game);
      } else if
         [...]
    }
    state.mouseDown(e, x, y);
  }
```

```java
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  state.mouseUp(e, x, y);
  state = theNullTool;
}
@Override
public void mouseDrag(MouseEvent e, int x, int y) {
  state.mouseDrag(e, x, y);
}
@Override
public void mouseMove(MouseEvent e, int x, int y) {
  state.mouseMove(e, x, y);
}
```

AARHUS UNIVERSITET

Development Process

How to implement this in
a *small steps* way?

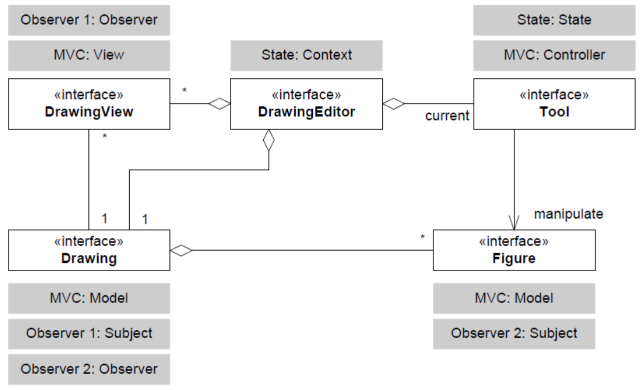
The Issue

- If we…
  - Implement GUI to Domain code first – there is no visual feedback
    - As there are no ‘Domain to GUI’ code in place to update Gfx ☹
  - Implement the Domain to GUI code first – then nothing happens
    - As there is no way to force any state changes ☹
      - (no game mutator calls)
- Solution: Need something to break the dependency
- **FakeObject: Replacement object that is a lightweight implementation of near-realistic behavior**

Domain To GUI

- We start in this end…

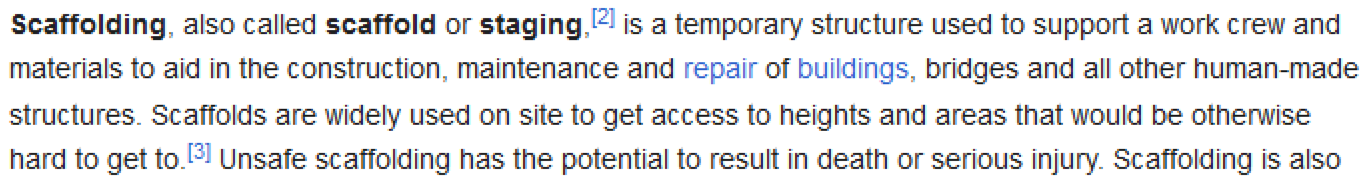
«interface»
Game



- … and must *fake* the state changes on game
  - Some *other means* of calling
    - game.playCard(…)
    - game.attackHero(…)
  - … than using the UI itself
- **We do it using a 'FakeObject' tool,** constructed just for this particular development task (ala a Junit test case)

Scaffolding

- Scaffolding:

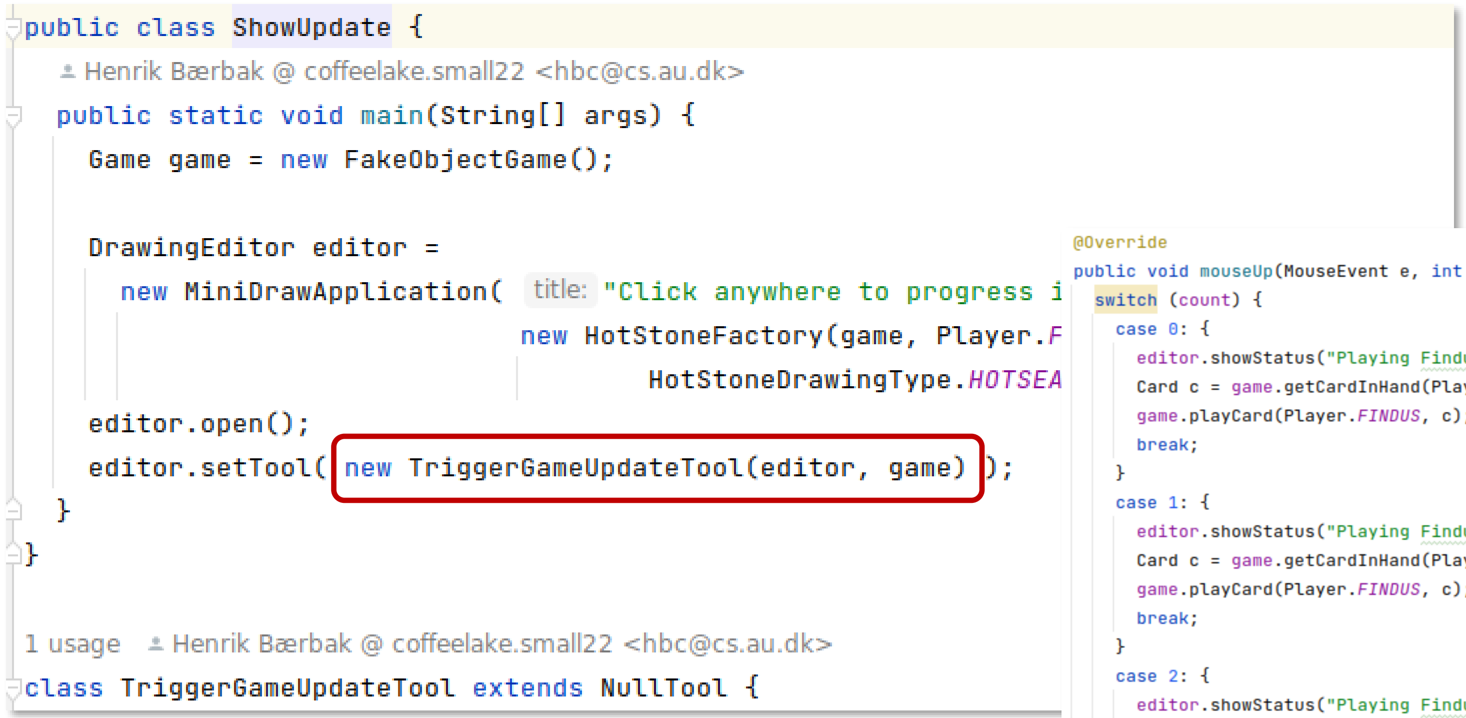


Scaffolding, also called **scaffold** or **staging**,[2] is a temporary structure used to support a work crew and materials to aid in the construction, maintenance and repair of buildings, bridges and all other human-made structures. Scaffolds are widely used on site to get access to heights and areas that would be otherwise hard to get to.[3] Unsafe scaffolding has the potential to result in death or serious injury. Scaffolding is also

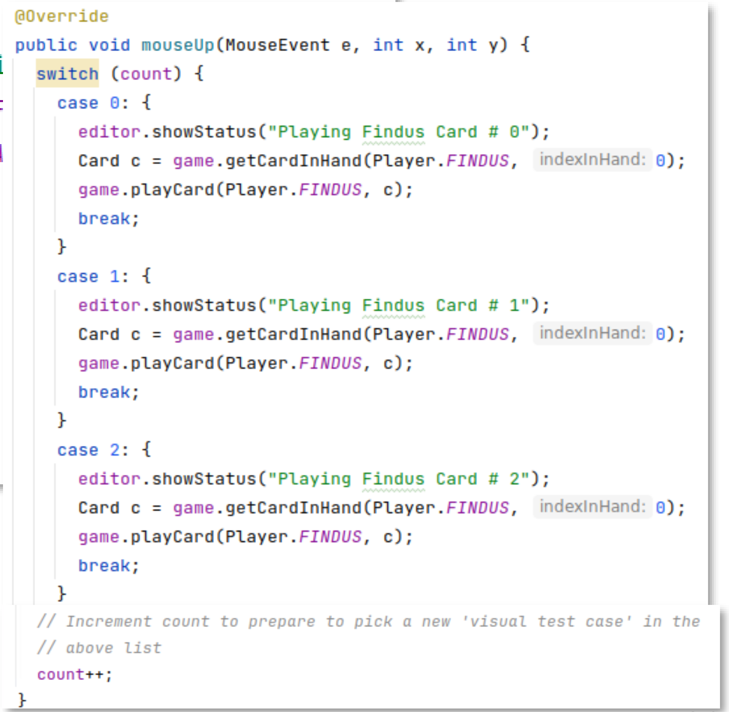
- **Scaffolding code:**
  - Code to help us build the production code, not part of the production itself
    - A manual 'JUnit' pendant…

AARHUS UNIVERSITET

Scaffolding Tool



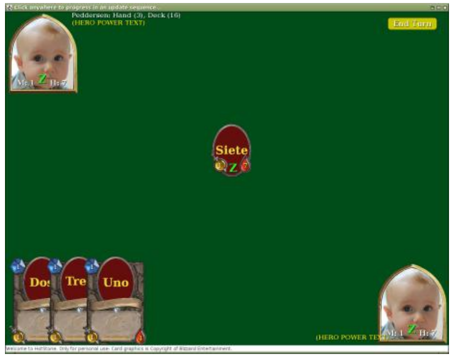
```
public class ShowUpdate {
  ≗ Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public static void main(String[] args) {
    Game game = new FakeObjectGame();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to progress i
                                new HotStoneFactory(game, Player.F
                                    HotStoneDrawingType.HOTSEA
    editor.open();
    editor.setTool( new TriggerGameUpdateTool(editor, game) );
  }
}

1 usage  ≗ Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
class TriggerGameUpdateTool extends NullTool {
```

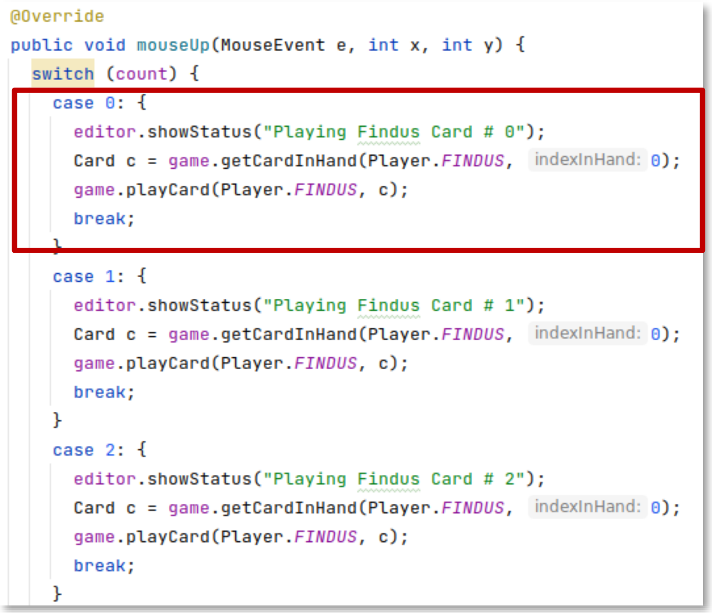


```
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  switch (count) {
    case 0: {
      editor.showStatus("Playing Findus Card # 0");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 1: {
      editor.showStatus("Playing Findus Card # 1");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 2: {
      editor.showStatus("Playing Findus Card # 2");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
  // Increment count to prepare to pick a new 'visual test case' in the
  // above list
  count++;
}
```

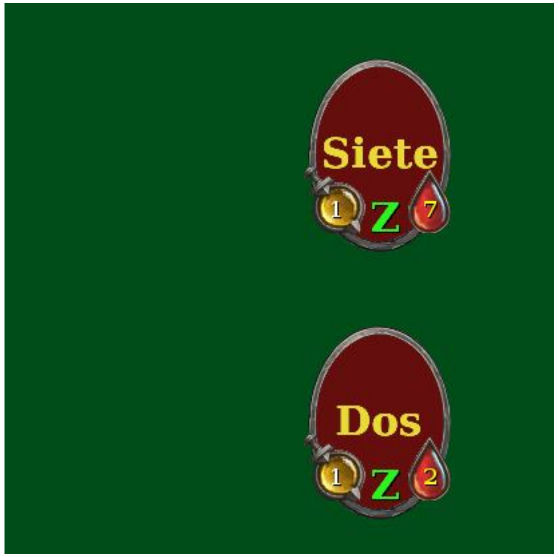
Demo

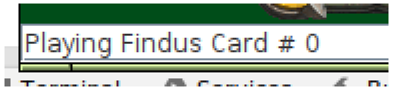
- Click First time
  - = first test case





```
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  switch (count) {
    case 0: {
      editor.showStatus("Playing Findus Card # 0");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 1: {
      editor.showStatus("Playing Findus Card # 1");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 2: {
      editor.showStatus("Playing Findus Card # 2");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
```





Playing Findus Card # 0

AARHUS UNIVERSITET

Why Does This Work?

I am sure it was called

```
Game game = new FakeObjectGame();
```

```
FakeObjectGame:: playCard(FINDUS, Dos) called...
```

```java
public Status playCard(Player who, Card card) {
  System.out.println(" FakeObjectGame:: playCard(" + who + ", " + card.getName() + ") called...");
  findusHand.remove(card);
  findusField.add(card);
  observerHandler.notifyPlayCard(who, card);
  return Status.OK;
}
```

```java
public void notifyPlayCard(Player who, Card card) {
  observerList
          .forEach( gameObserver -> gameObserver.onCardPlay(who, card) );
}
```

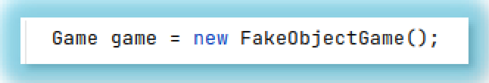
```java
public class HotStoneDrawing implements Drawing, GameObserver {
```

```java
public void onCardPlay(Player who, Card card) {
  addMessage("" + who + " plays " + card.getName() + ".");
  // TODO: Add another message if the card has an effect

  // As this direct mutator call has known side effects which are
  // not represented by the indirect observer notifications, the
  // card/minion updates are effected here: Remove the card figure
  // and replace it with a minion figure.
  removeActorAndUpdateMapping(card);
  createActorAndUpdateMapping(card, HotStoneFigureType.MINION_FIGURE);

  refreshField(who);

  opponentSummary.setText(computeHeroSummary(
          Utility.computeOpponent(playerShown)));
}
```

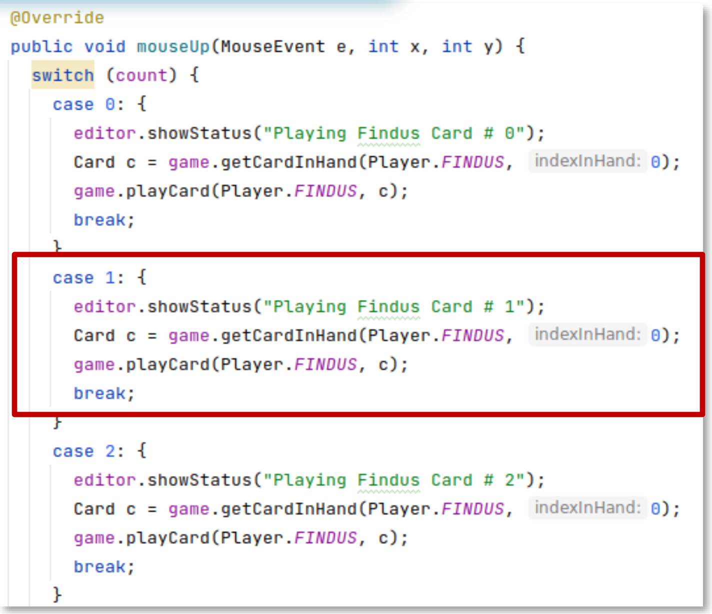
AARHUS UNIVERSITET

Demo

- Click Second time
  - = second test case



```
Game game = new FakeObjectGame();
```



```
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  switch (count) {
    case 0: {
      editor.showStatus("Playing Findus Card # 0");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 1: {
      editor.showStatus("Playing Findus Card # 1");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 2: {
      editor.showStatus("Playing Findus Card # 2");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
```

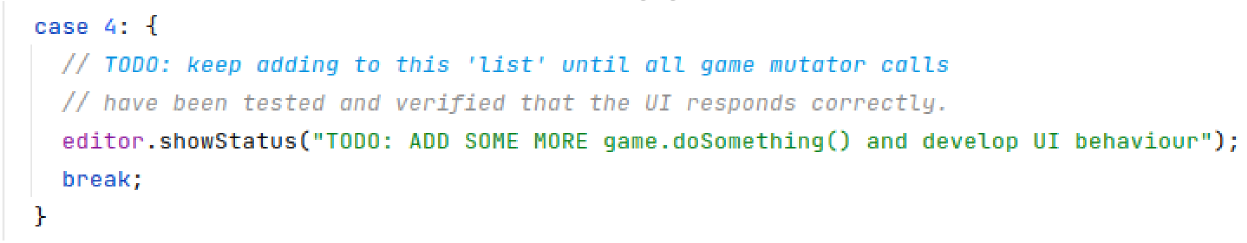
… and see the right thing happen *visually*

And so on…

AARHUS UNIVERSITET

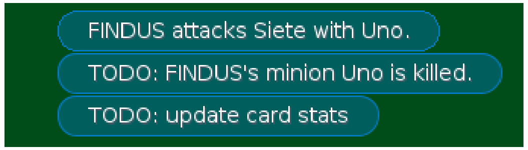
The Point

- ShowUpdate program is a *visual test program*
  - To Test Drive the full HotStoneDrawing implementation
  - To *add test cases*, put tests into the TriggerGameUpdate tool



```
case 4: {
  // TODO: keep adding to this 'list' until all game mutator calls
  // have been tested and verified that the UI responds correctly.
  editor.showStatus("TODO: ADD SOME MORE game.doSomething() and develop UI behaviour");
  break;
}
```

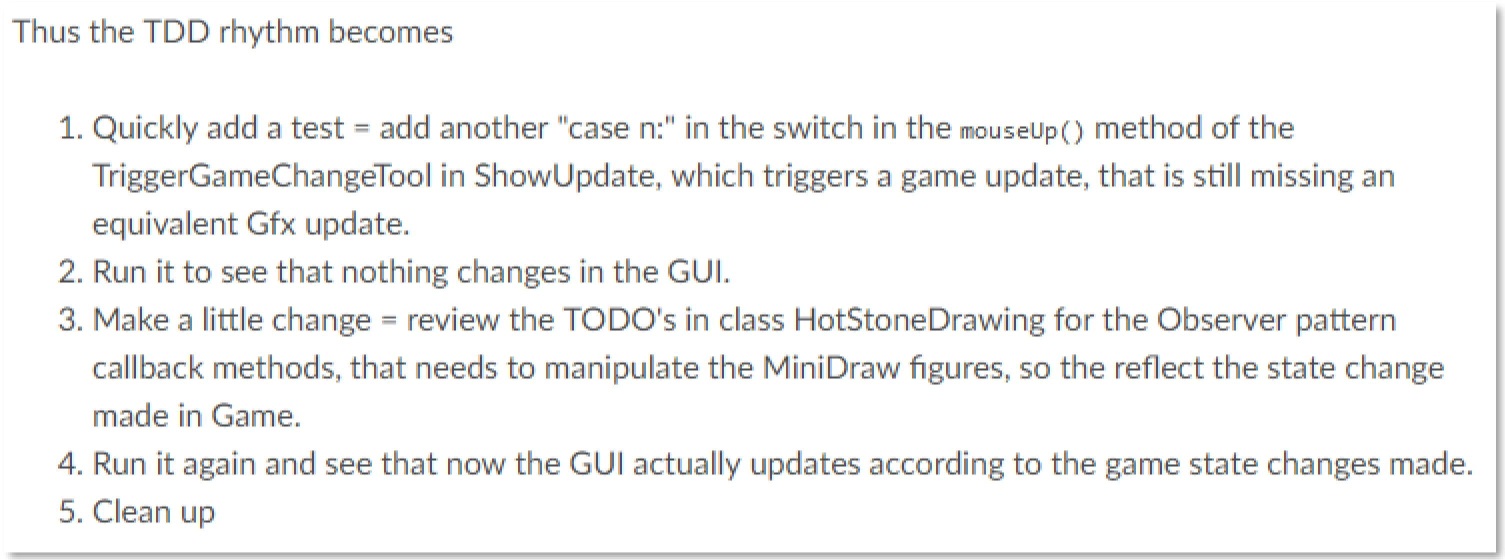
  - Add case 5, 6, … until all mutators in game can be called and the Gfx is updated correctly!



FINDUS attacks Siete with Uno.
TODO: FINDUS's minion Uno is killed.
TODO: update card stats

AARHUS UNIVERSITET

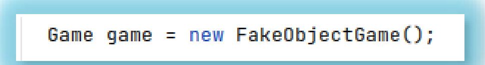
Alas the Rhythm


Thus the TDD rhythm becomes

1. Quickly add a test = add another "case n:" in the switch in the `mouseUp()` method of the TriggerGameChangeTool in ShowUpdate, which triggers a game update, that is still missing an equivalent Gfx update.
2. Run it to see that nothing changes in the GUI.
3. Make a little change = review the TODO's in class HotStoneDrawing for the Observer pattern callback methods, that needs to manipulate the MiniDraw figures, so the reflect the state change made in Game.
4. Run it again and see that now the GUI actually updates according to the game state changes made.
5. Clean up

AARHUS UNIVERSITET

Test Double or Not?



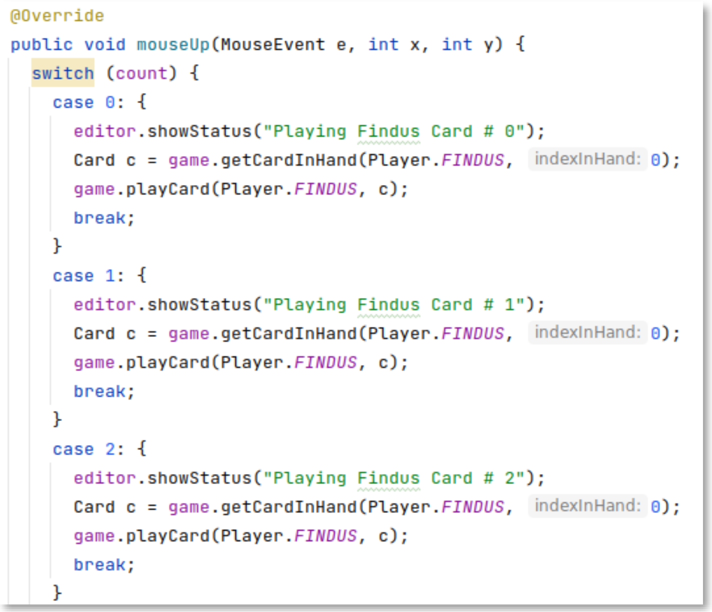
```
Game game = new FakeObjectGame();
```

- *You can do this using Test Doubles*
  - A *FakeObjectGame* that mimic a real game but with stubbed/canned behavior
  - Experience: It becomes almost like developing AlphaStone all over again…
    - Laborious and even ripe for coding errors in the FakeObject ☹
- *You can do this using a simple HotStone variant*
  - Like e.g. AlphaStone
  - Argument: *At this point our AlphaStone should be pretty '… code that works' without defects and thus may serve as basis for incrementally developing our GUI*

AARHUS UNIVERSITET

But (1)...

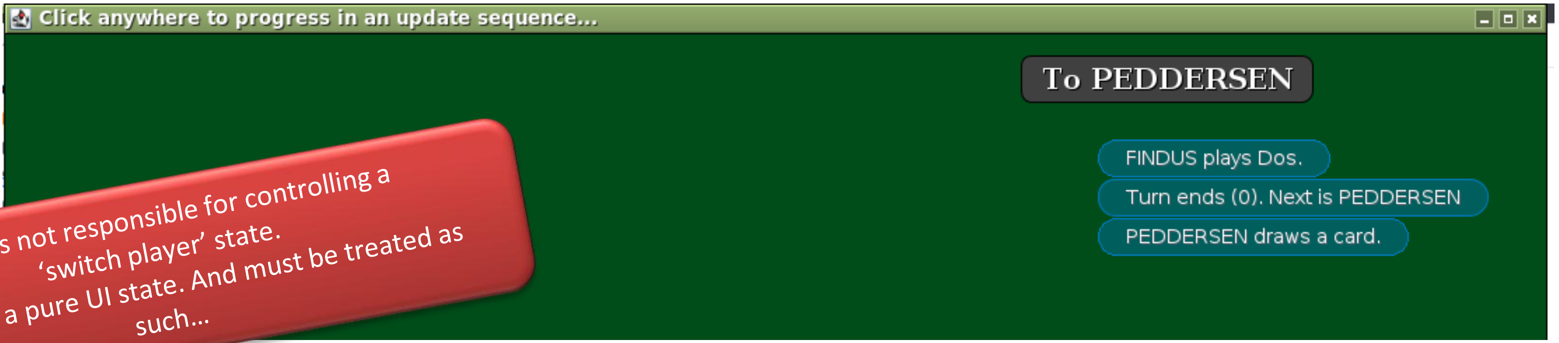
- There are some trap if you change to, say, AlphaStone
- First:
  - AlphaStone Hero has tree mana
  - And cards (Tres, Dos, Uno)
- Thus – is this ‘test sequence’ valid?
  - Answer: No
    - Argue why?
- Morale: Change the sequence!



```
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  switch (count) {
    case 0: {
      editor.showStatus("Playing Findus Card # 0");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 1: {
      editor.showStatus("Playing Findus Card # 1");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
    case 2: {
      editor.showStatus("Playing Findus Card # 2");
      Card c = game.getCardInHand(Player.FINDUS,  indexInHand: 0);
      game.playCard(Player.FINDUS, c);
      break;
    }
```

But (2)…

- Some visual tests require Peddersen to do stuff
  - We cannot test ‘attackCard()’s visual behavior unless Peddersen has some minions on the field, right?
- Easy (?)
  - Just call endTurn()… Ups?!?
    - How do get rid of that state of the UI???

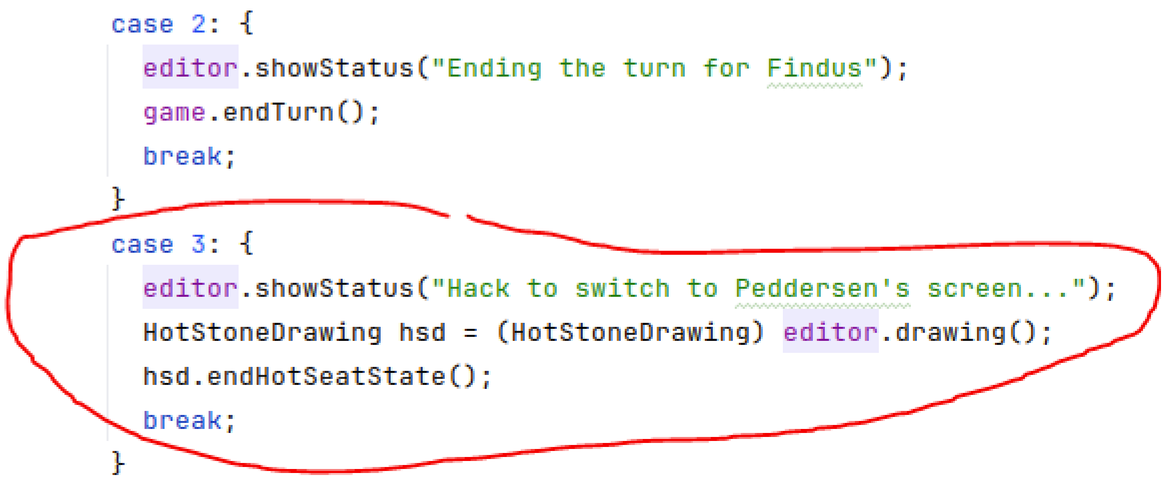

Click anywhere to progress in an update sequence...
To PEDDERSEN
FINDUS plays Dos.
Turn ends (0). Next is PEDDERSEN
PEDDERSEN draws a card.

Game is not responsible for controlling a ‘switch player’ state.
Thus this is a pure UI state. And must be treated as such…

AARHUS UNIVERSITET

But (2) …

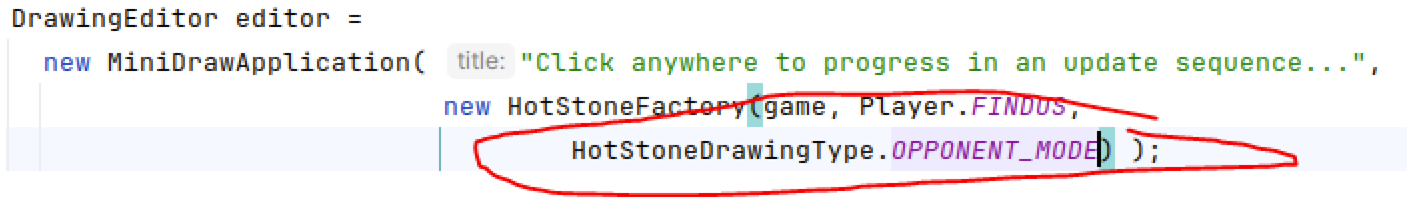
- Our TriggerUpdateTool must therefore handle this…
- The trick is:
  - Force the UI state change directly from our scaffolding code…



```
case 2: {
  editor.showStatus("Ending the turn for Findus");
  game.endTurn();
  break;
}
case 3: {
  editor.showStatus("Hack to switch to Peddersen's screen...");
  HotStoneDrawing hsd = (HotStoneDrawing) editor.drawing();
  hsd.endHotSeatState();
  break;
}
```

But (2) Alternative…

- An alternative route is to use HotStoneDrawing in its ‘opponent mode’
  - Which is designed for remote play
    - The UI is *always tied to one specific player*
      - Here it is Findus.

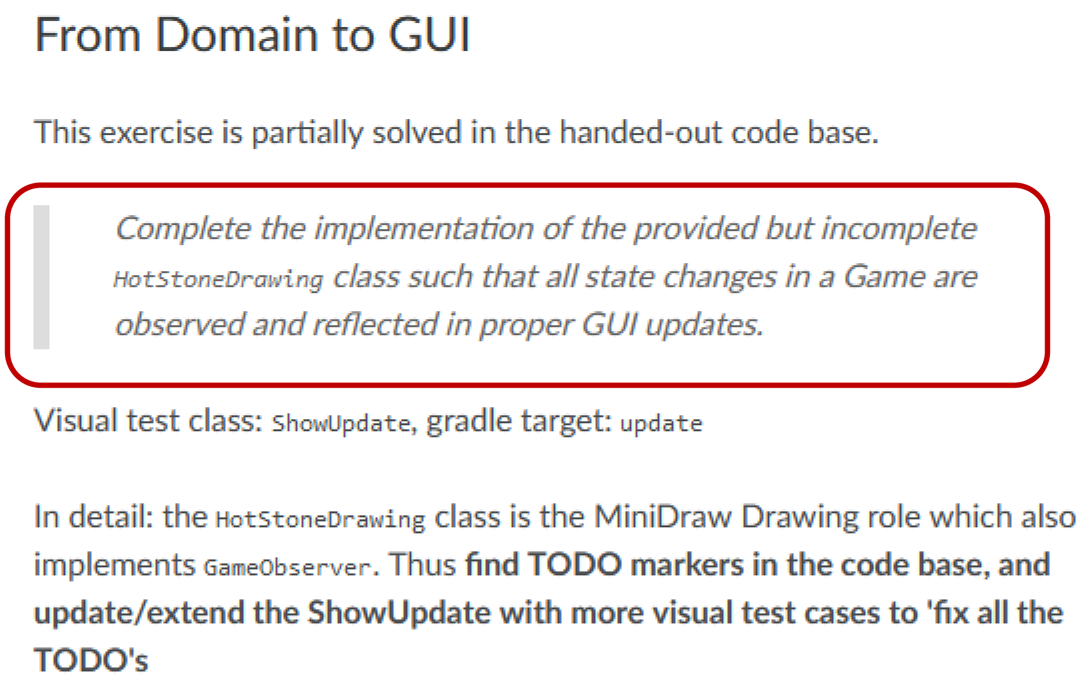


```
DrawingEditor editor =
  new MiniDrawApplication( title: "Click anywhere to progress in an update sequence...",
                           new HotStoneFactory(game, Player.FINDUS,
                               HotStoneDrawingType.OPPONENT_MODE) );
```

  - Now a ‘case n:’ state can:
    - endTurn(); (do some Peddersen mutator calls); endTurn();
    - *Remember – you must end in a state where Findus is in turn!*

Mandatory: Domain to GUI

- The first MiniDraw related exercise on Iteration 8



From Domain to GUI

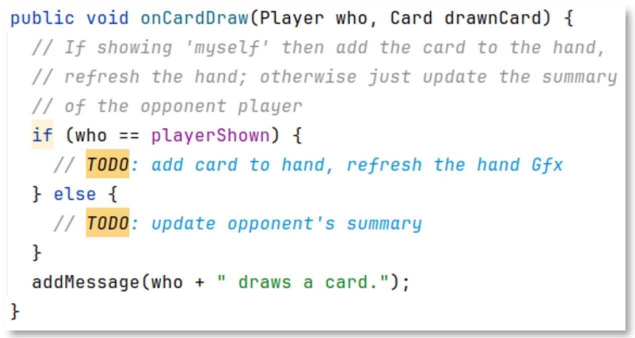
This exercise is partially solved in the handed-out code base.

> *Complete the implementation of the provided but incomplete `HotStoneDrawing` class such that all state changes in a Game are observed and reflected in proper GUI updates.*

Visual test class: `ShowUpdate`, gradle target: `update`

In detail: the `HotStoneDrawing` class is the MiniDraw Drawing role which also implements `GameObserver`. Thus **find TODO markers in the code base, and update/extend the ShowUpdate with more visual test cases to 'fix all the TODO's**

Look for TODO comments in the code.



```
public void onCardDraw(Player who, Card drawnCard) {
  // If showing 'myself' then add the card to the hand,
  // refresh the hand; otherwise just update the summary
  // of the opponent player
  if (who == playerShown) {
    // TODO: add card to hand, refresh the hand Gfx
  } else {
    // TODO: update opponent's summary
  }
  addMessage(who + " draws a card.");
}
```

GUI to Domain

GUI to Domain

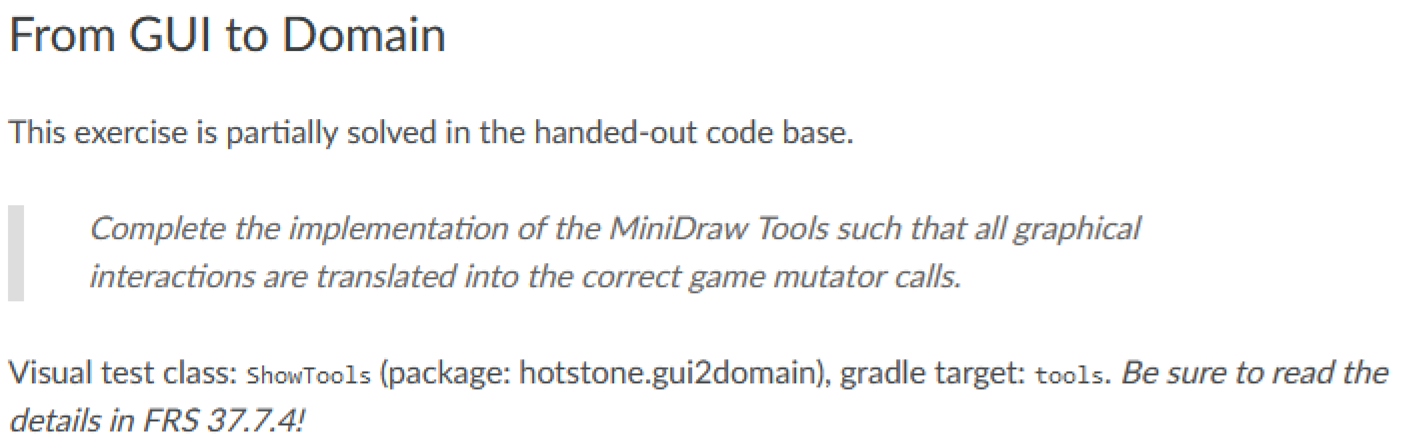
- ShowTools is a *visual test program*

```
public class ShowTools {
  public static void main(String[] args) {
    // TODO: Probably replace with a real but simple HotStone variant
    Game game = new FakeObjectGame();

    DrawingEditor editor =
            new MiniDrawApplication( "Test-Driven Dev of Tools",
                    new HotStoneFactory(game, Player.FINDUS,
                            HotStoneDrawingType.HOTSEAT_MODE) );
    editor.open();
    editor.setTool(new HotSeatStateTool(editor, game));
  }
}
```

- Process
  - Pick a ‘user action’ = tool to implement
  - Develop Xtool (+ extend HotSeatStateTool) for that until OK
  - Loop until all four user actions are done…

Mandatory: GUI to Domain

- That is



From GUI to Domain

This exercise is partially solved in the handed-out code base.

> *Complete the implementation of the MiniDraw Tools such that all graphical interactions are translated into the correct game mutator calls.*

Visual test class: `ShowTools` (package: hotstone.gui2domain), gradle target: `tools`. *Be sure to read the details in FRS 37.7.4!*

- Again, use a real game implementation rather than my FakeObject game
  - Exercise: Why do the provided code use a FakeObject Game?

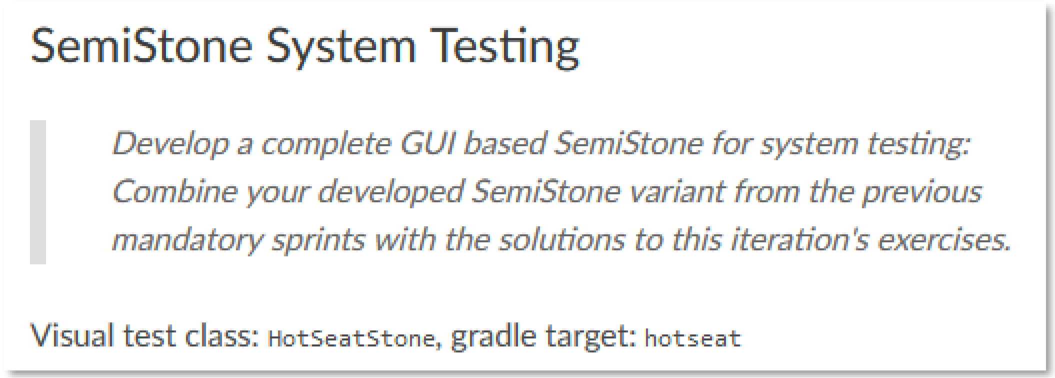
AARHUS UNIVERSITET

System Testing

Play the Game.
Finally!

System Testing

- And – Ta daa!



SemiStone System Testing

> *Develop a complete GUI based SemiStone for system testing: Combine your developed SemiStone variant from the previous mandatory sprints with the solutions to this iteration's exercises.*

Visual test class: `HotSeatStone`, gradle target: `hotseat`

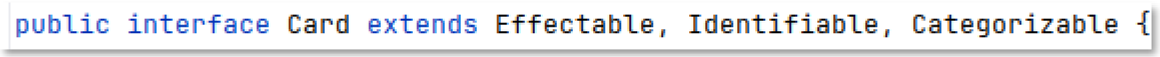
- Play any of the variants…

AARHUS UNIVERSITET

Two SubSystems
=
Two set of Terms/Objects

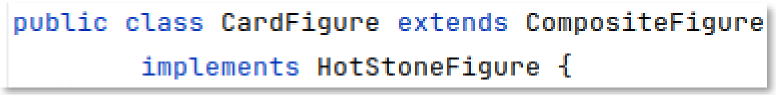
Domain and GUI Terms

- Domain / Game speaks in terms of Card, Hero, Player
  - To represent the **roles** of a card, a hero, …



```
public interface Card extends Effectable, Identifiable, Categorizable {
```

- GUI / MiniDraw speaks in terms of CardFigure, HeroFigure, …
  - To represent the **roles** of a card, a hero

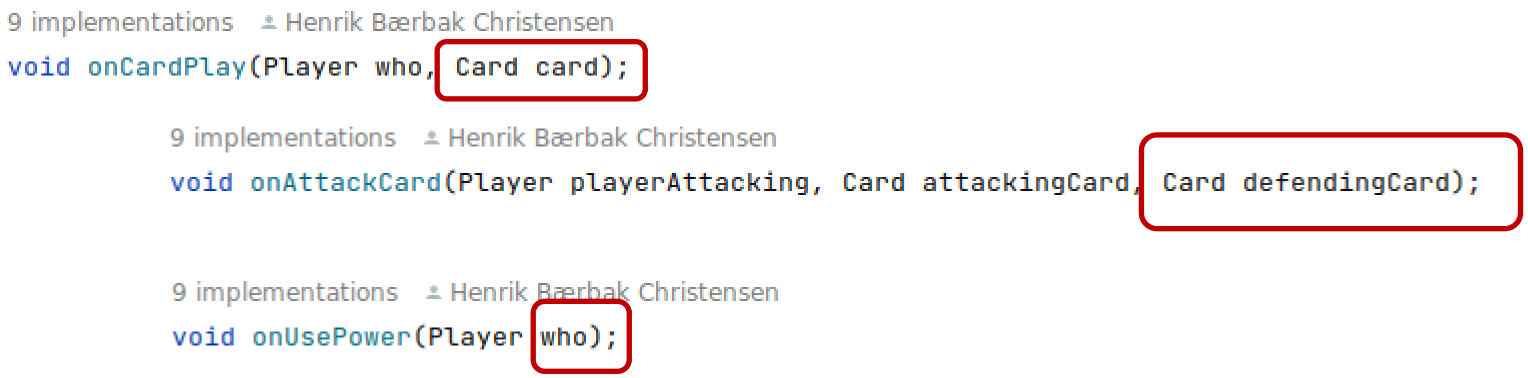


```
public class CardFigure extends CompositeFigure
        implements HotStoneFigure {
```

- *We need to translate between one and the other when the two subsystems interact…*

Coupling Domain and Gfx

- The Observer onX() calls “speaks in domain terms”

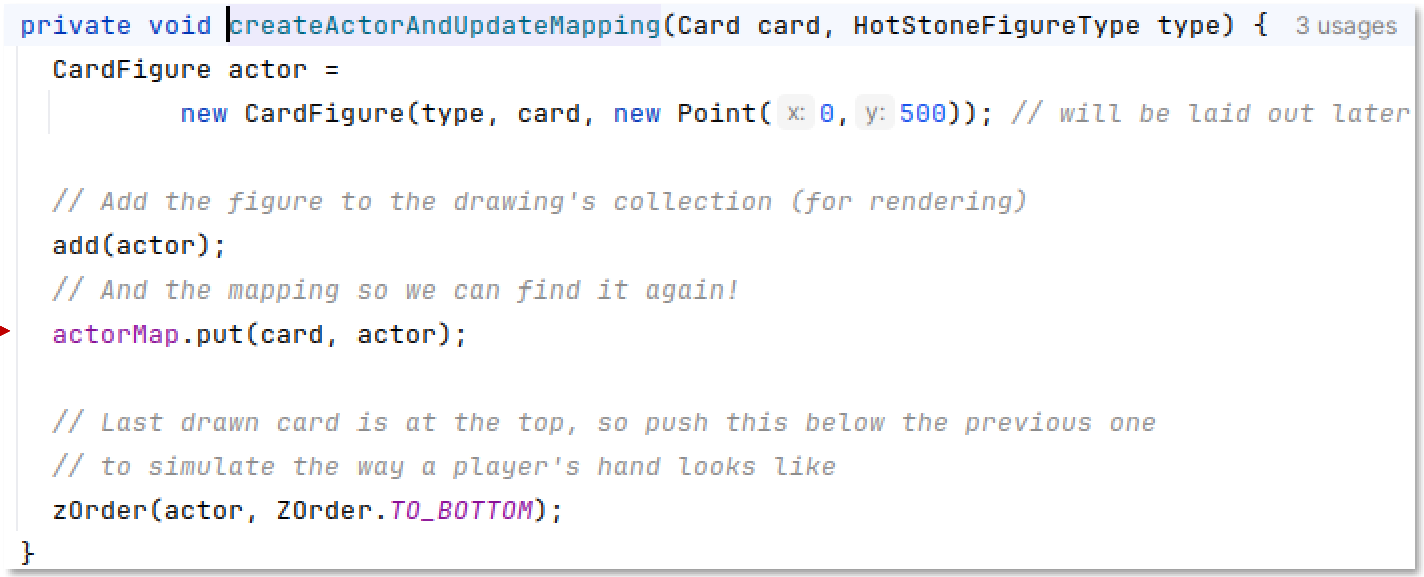


```
9 implementations   Henrik Bærbak Christensen
void onCardPlay(Player who, Card card);

    9 implementations   Henrik Bærbak Christensen
    void onAttackCard(Player playerAttacking, Card attackingCard, Card defendingCard);

    9 implementations   Henrik Bærbak Christensen
    void onUsePower(Player who);
```

- But these have to be translated into MiniDraw Gfx equivalents: A card is drawn as a CardFigure…
  - Solution: **Drawing keeps a mapping between the two**

Coupling Domain and Gfx

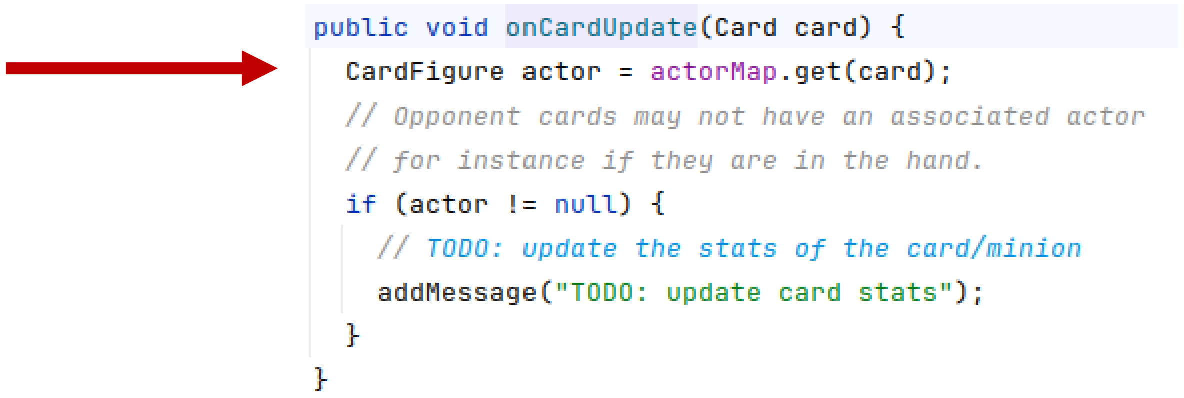
- In
```java
public class HotStoneDrawing implements Drawing, GameObserver
```
- A simple
```java
private Map<Card, CardFigure> actorMap;
```
does the trick



```java
private void createActorAndUpdateMapping(Card card, HotStoneFigureType type) {  3 usages
  CardFigure actor =
          new CardFigure(type, card, new Point( x: 0, y: 500)); // will be laid out later

  // Add the figure to the drawing's collection (for rendering)
  add(actor);
  // And the mapping so we can find it again!
  actorMap.put(card, actor);

  // Last drawn card is at the top, so push this below the previous one
  // to simulate the way a player's hand looks like
  zOrder(actor, ZOrder.TO_BOTTOM);
}
```

Coupling Domain and Gfx

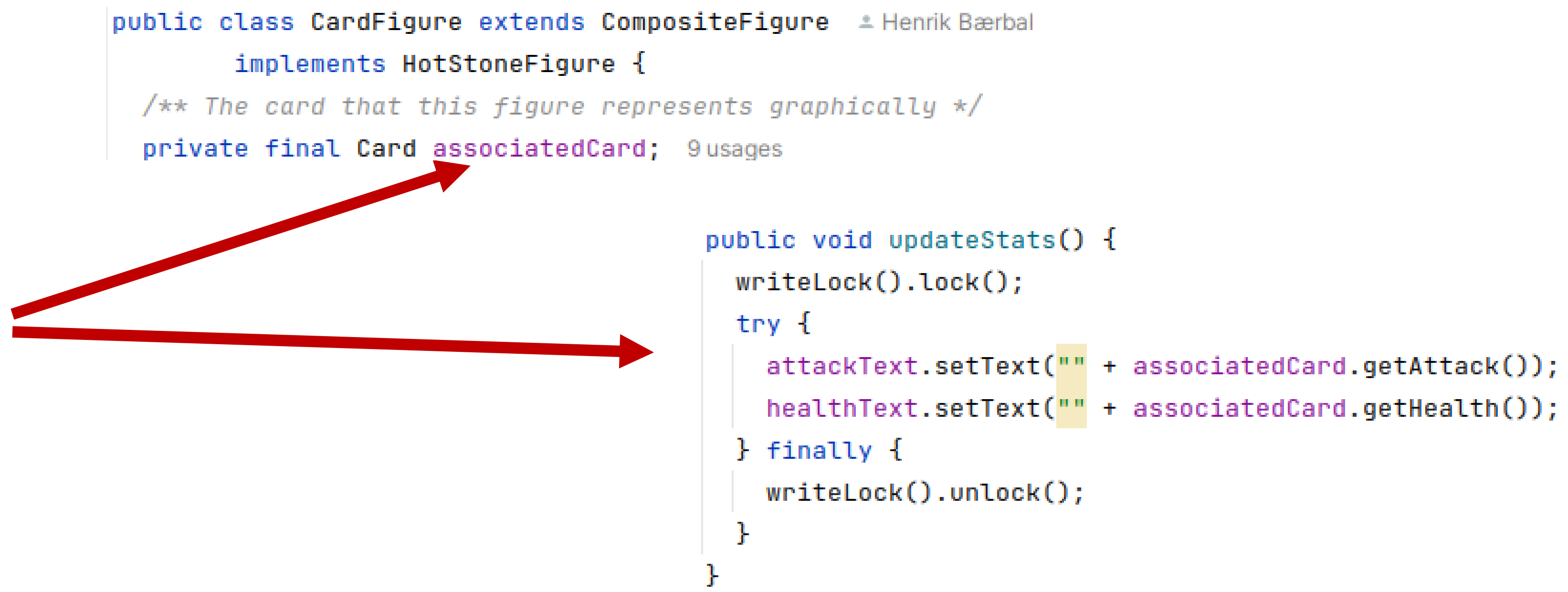
- Which allows us to handle onCardUpdate(), simply by looking up that card



```
public void onCardUpdate(Card card) {
  CardFigure actor = actorMap.get(card);
  // Opponent cards may not have an associated actor
  // for instance if they are in the hand.
  if (actor != null) {
    // TODO: update the stats of the card/minion
    addMessage("TODO: update card stats");
  }
}
```

And The Other Way

- We need the link the other way as well
  - The Figure needs access to its associated Card/minion

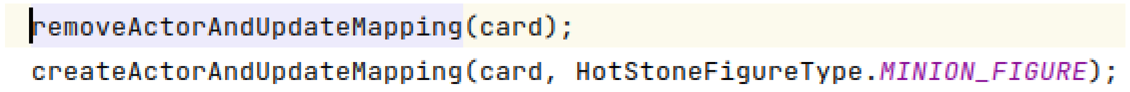


```java
public class CardFigure extends CompositeFigure  Henrik Bærbal
        implements HotStoneFigure {
  /** The card that this figure represents graphically */
  private final Card associatedCard;  9 usages
```

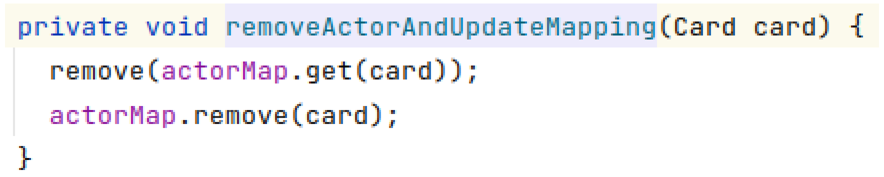
```java
public void updateStats() {
  writeLock().lock();
  try {
    attackText.setText("" + associatedCard.getAttack());
    healthText.setText("" + associatedCard.getHealth());
  } finally {
    writeLock().unlock();
  }
}
```

Liability of a Mapping

- The big downside of a mapping !
  - Your code has to ensure they do not get out of sync!
    - Looking up a Figure which is not there ☹
- Solution is
  - Uncle Bob: *One Level Of Abstraction*
    - *Never manipulate map directly, use private methods*



```
removeActorAndUpdateMapping(card);
createActorAndUpdateMapping(card, HotStoneFigureType.MINION_FIGURE);
```



```
private void removeActorAndUpdateMapping(Card card) {
  remove(actorMap.get(card));
  actorMap.remove(card);
}
```

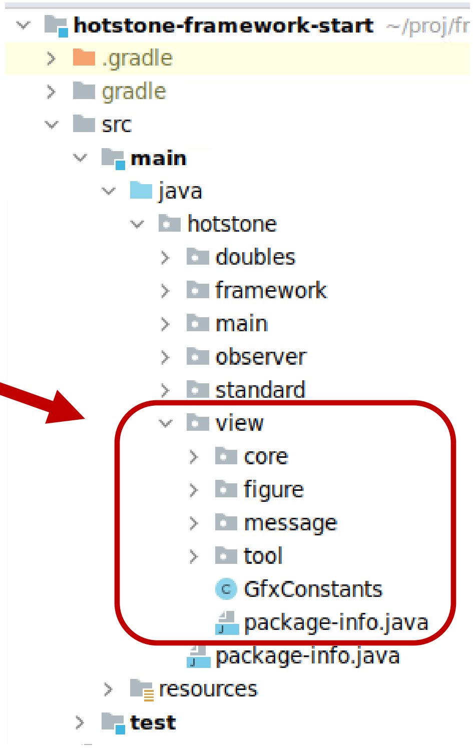
  - And **lots of testing!**

Module View

Organization of interfaces/classes

The Provided Code…

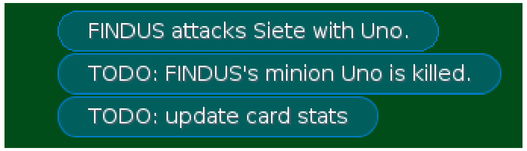
- Provided Zip adds a lot of new code to your HotStone
  - Merge carefully, *temporary stubs for StandardGame*, may overwrite all your work!
  - Do it on a branch!!! Make a backup
- New UI specific packages



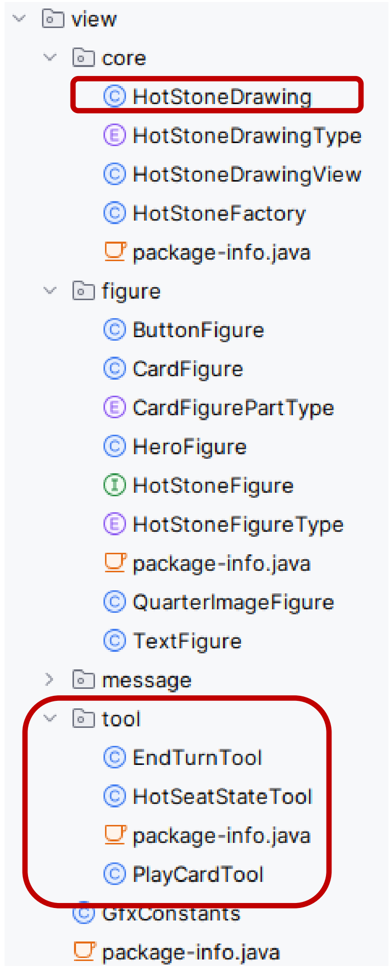
```
hotstone-framework-start ~/proj/fr
  .gradle
  gradle
  src
    main
      java
        hotstone
          doubles
          framework
          main
          observer
          standard
          view
            core
            figure
            message
            tool
            GfxConstants
            package-info.java
          package-info.java
      resources
    test
```

Package Contents

- Core
  - MiniDraw 'Drawing' and 'View' roles
    - Partially implemented for you
- Figure
  - MiniDraw 'Figure's
- Message
  - Aux roles
- Tool
  - MiniDraw 'Tool' roles
    - Partially provided for you
- GfxConstants *All layout stored there*



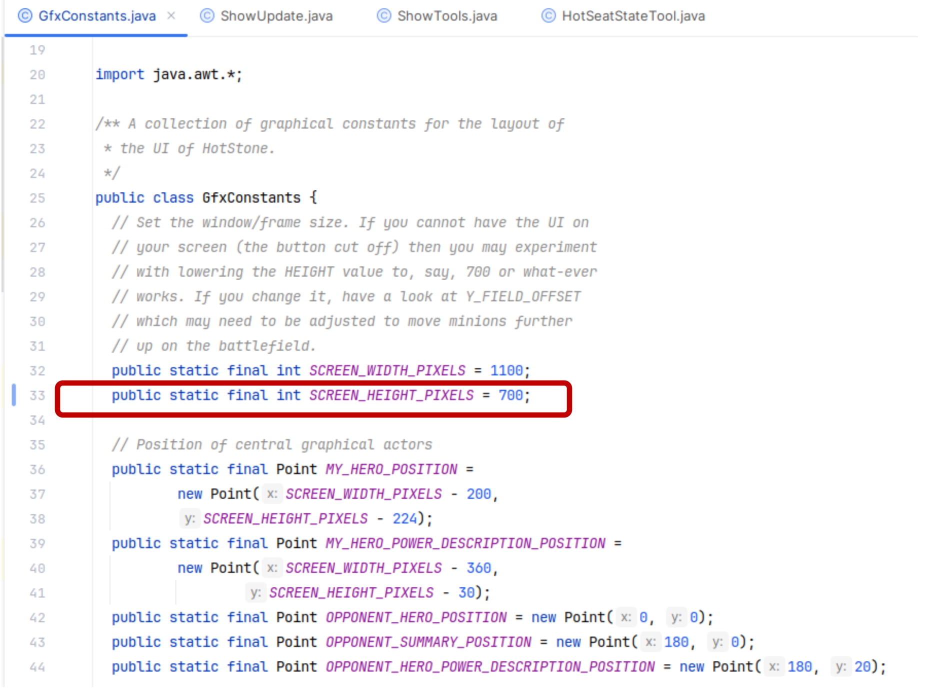
FINDUS attacks Siete with Uno.
TODO: FINDUS's minion Uno is killed.
TODO: update card stats



```
view
  core
    HotStoneDrawing
    HotStoneDrawingType
    HotStoneDrawingView
    HotStoneFactory
    package-info.java
  figure
    ButtonFigure
    CardFigure
    CardFigurePartType
    HeroFigure
    HotStoneFigure
    HotStoneFigureType
    package-info.java
    QuarterImageFigure
    TextFigure
  message
  tool
    EndTurnTool
    HotSeatStateTool
    package-info.java
    PlayCardTool
  GfxConstants
  package-info.java
```

GfxConstants

- Controlling (most) graphical layouts
  - The Peter Bøgh projectors are 1600x900 pixels so I have to adjust the
    - SCREEN_HEIGHT_PIXEL
- Play around to find the most pleasing size on your machine…
- (No viewpoint in MiniDraw, sorry…)



```java

import java.awt.*;

/** A collection of graphical constants for the layout of
 * the UI of HotStone.
 */
public class GfxConstants {
  // Set the window/frame size. If you cannot have the UI on
  // your screen (the button cut off) then you may experiment
  // with lowering the HEIGHT value to, say, 700 or what-ever
  // works. If you change it, have a look at Y_FIELD_OFFSET
  // which may need to be adjusted to move minions further
  // up on the battlefield.
  public static final int SCREEN_WIDTH_PIXELS = 1100;
  public static final int SCREEN_HEIGHT_PIXELS = 700;

  // Position of central graphical actors
  public static final Point MY_HERO_POSITION =
          new Point( x: SCREEN_WIDTH_PIXELS - 200,
           y: SCREEN_HEIGHT_PIXELS - 224);
  public static final Point MY_HERO_POWER_DESCRIPTION_POSITION =
          new Point( x: SCREEN_WIDTH_PIXELS - 360,
                  y: SCREEN_HEIGHT_PIXELS - 30);
  public static final Point OPPONENT_HERO_POSITION = new Point( x: 0,  y: 0);
  public static final Point OPPONENT_SUMMARY_POSITION = new Point( x: 180,  y: 0);
  public static final Point OPPONENT_HERO_POWER_DESCRIPTION_POSITION = new Point( x: 180,  y: 20);
```

AARHUS UNIVERSITET

Happy Coding…