Com S 362
Object-Oriented Analysis & Design

Intro to the Project

Term Project

Thinking About Complex Problems

- Computer Solutions are a ***bad*** model.
  - Computers are our only general purpose tool – they can do anything, anyway – no constraint.
  - Computers and computer languages are too abstract and granular – no problem specific meaning, no help.
- Manual solutions are much more helpful – thus the idea of "domain models".

Thinking about Complex Problems

- Focus on decisions:
  - What decisions are required? can you break them into their smallest constituent parts?
  - what is the *absolute minimum* information required to make each decision?
  - what unique or separate work (if any) does it enable?
    - some decisions just enable other decisions.
- Examine what happens if the context is slightly different in one way or another
  - This helps clarify what is and what is not a part of a specific decision. It often helps you identify subliminal assumptions.

An Example
Thinking About Games

Assume you go to ‘game night’.

- What is the first decision you must make?
- What is the minimum you need to know about the environment/context to make this decision?
- What are the parts of the decision if someone has already set up each available game on a separate table?

Does this fit?

- (Decision) Select a game to play
- (work) Identify additional players if necessary and possible
- (Decision) Assign player roles/teams/positions
- (work) set up game resources (maybe done for you already)
- Begin play

Summary Description

- Game selection and provisioning
  - selection is input
- Match Configuration and initialization
  - Number of players and role is input
- Play
  - lots of player input

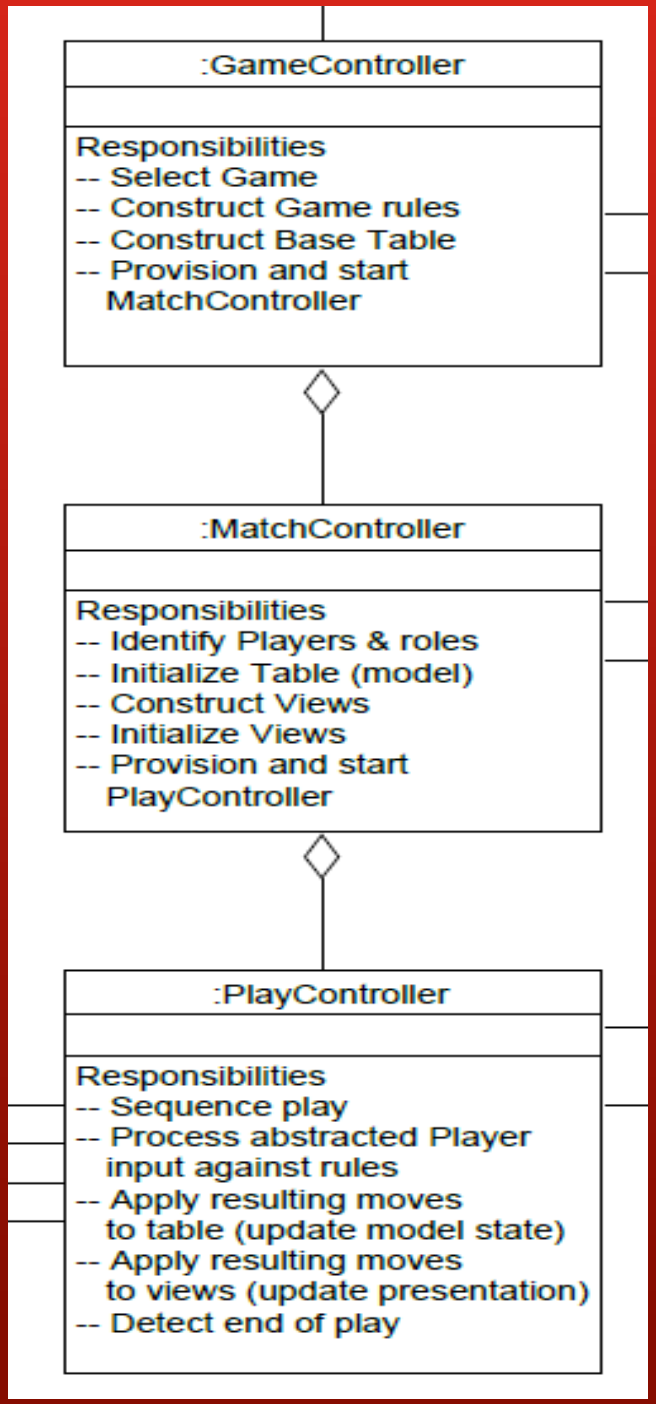
The presence of input at each level which influences different kinds of work suggests three “nested” controllers

Key Design Structure

Three Nested Controllers

- Game Controller
- Match Controller
- Play Controller

- Notice that the need for and the responsibilities of these controllers are the same in any game!

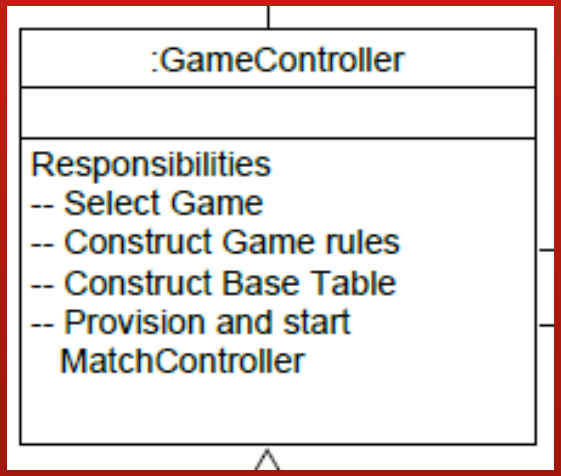


A very common structure

- Application
- Session
- Activity
- Transaction

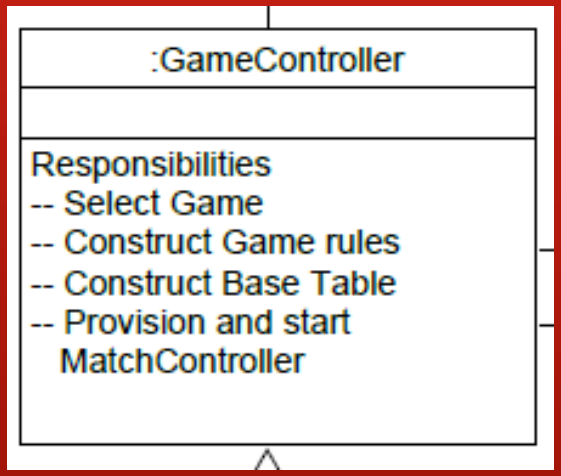
- Bootstrap
- Game
- Match
- Play

Hardwired Game Controller (Not Final Design)



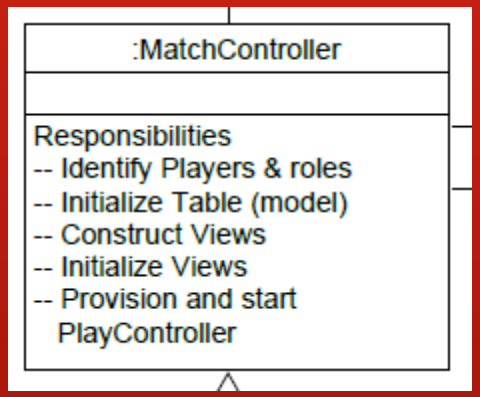

```
public class FiftyTwo {

    private InBoundQueue inQ;
    private RemoteTableGateway remote;

    public FiftyTwo (InBoundQueue inQ,
            RemoteTableGateway gateway){
        this.inQ = inQ;
        this.remote = gateway;
    }


    public void run (){
        Rules rules = new PickupRules();
        Table table = new TableBase();

        MatchController match =
            new MatchController(
              inQ, table, rules, remote
            );
        match.start();
    }
}
```

Generic Game Controller



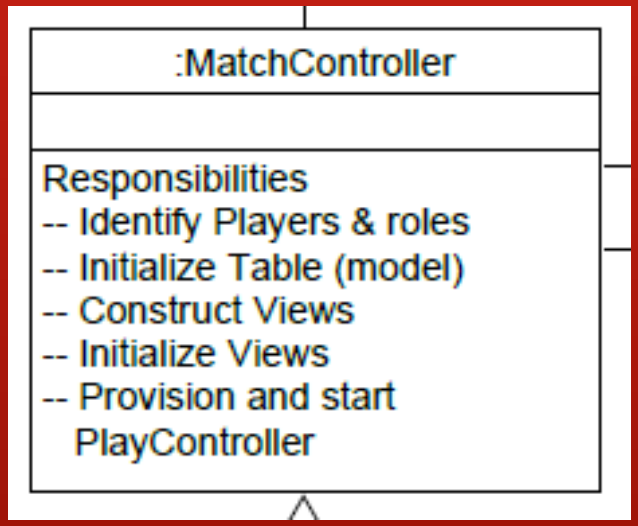

```
public void run() {
    Event e = null;
    while (!game.isSelected()) {
        try {
            e = inQ.take();
            ((SysEvent) e).accept(this, game);

        } catch (Exception ex) {
            // ...
        };
    }

    GameFactory factory = abstractFactory
            .getGameFactory(game.getSelection());
    Rules rules = factory.createRules();
    Table table = factory.createTable();
    inQ.pushBack(deferred);

    MatchController match = new MatchController(
            inQ, table, rules, remote, factory);
    match.start();
}
```

Hardwired Match Controller (Not Final Design)



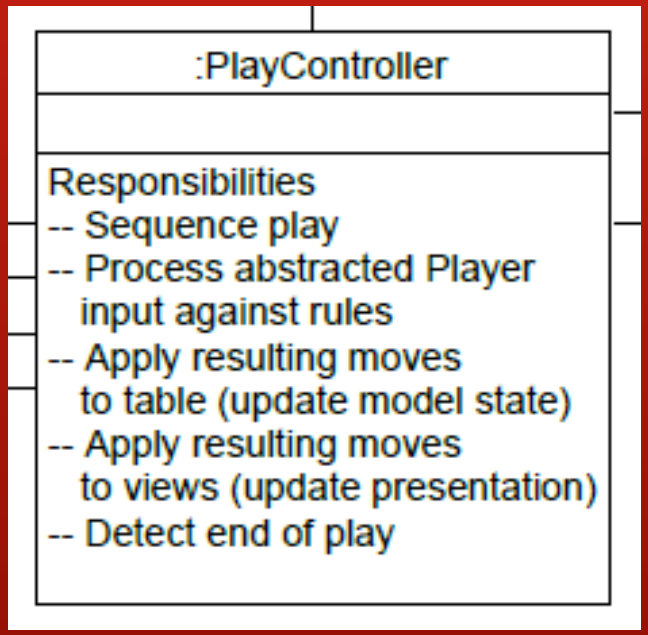

```
public void start(){
    //this is match setup ... it depends on whic
    //was selected. We initialize for a new matc
    //already selected game
    try {
        View  p1View = new P52PlayerView(1, remo
        views.add(p1View); // might be more or f

        p1View.send(new SetupTable());
        Player player = new PickupPlayer(1); //

        // initialize the local model for Pu52 m
        table.apply(new PickupInitCmd());
        p1View.apply(new PickupInitCmd());

        PlayController mainloop = new PlayContro
        mainloop.play(table, player, views);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
```

Generic Match Controller




```
public void start() {
    Event e = null;
    while (!table.partiesReady()) {
        try {
            e = inQ.take();
            Move cmd = rules.eval(
                    e, table, table.getCurrentPlayer());
            cmd.apply(table);
            cmd.apply(views);
        } catch (Exception ex) {
            // ...
        }
    }

    Move initCmd = rules.eval(
            new InitGameEvent(), table, null);
    initCmd.apply(table);
    initCmd.apply(views);

    PlayController mainloop =
            new PlayController(inQ, rules);
    mainloop.play(table, table.getCurrentPlayer(), views);
}
```

Play Controller




```
public class PlayController {

    private InBoundQueue inQ;
    private Rules rules;

    public PlayController (InBoundQueue inQ,
            Rules rules)
    {
        this.inQ = inQ;
        this.rules = rules;
    }

    public void play(Table table,
            Player player, List<View> views){
        Event nextE;
        View p1View = views.get(0);
        try {

            while ((nextE = inQ.take()) != null){
                Move move = rules
                    .eval(nextE, table, player);
                table.apply(move);
                p1View.apply(move);
            }
        } catch (InterruptedException e){
            System.err.println("Play terminated on e
            // clean up for next match?
        }

    }

}
```

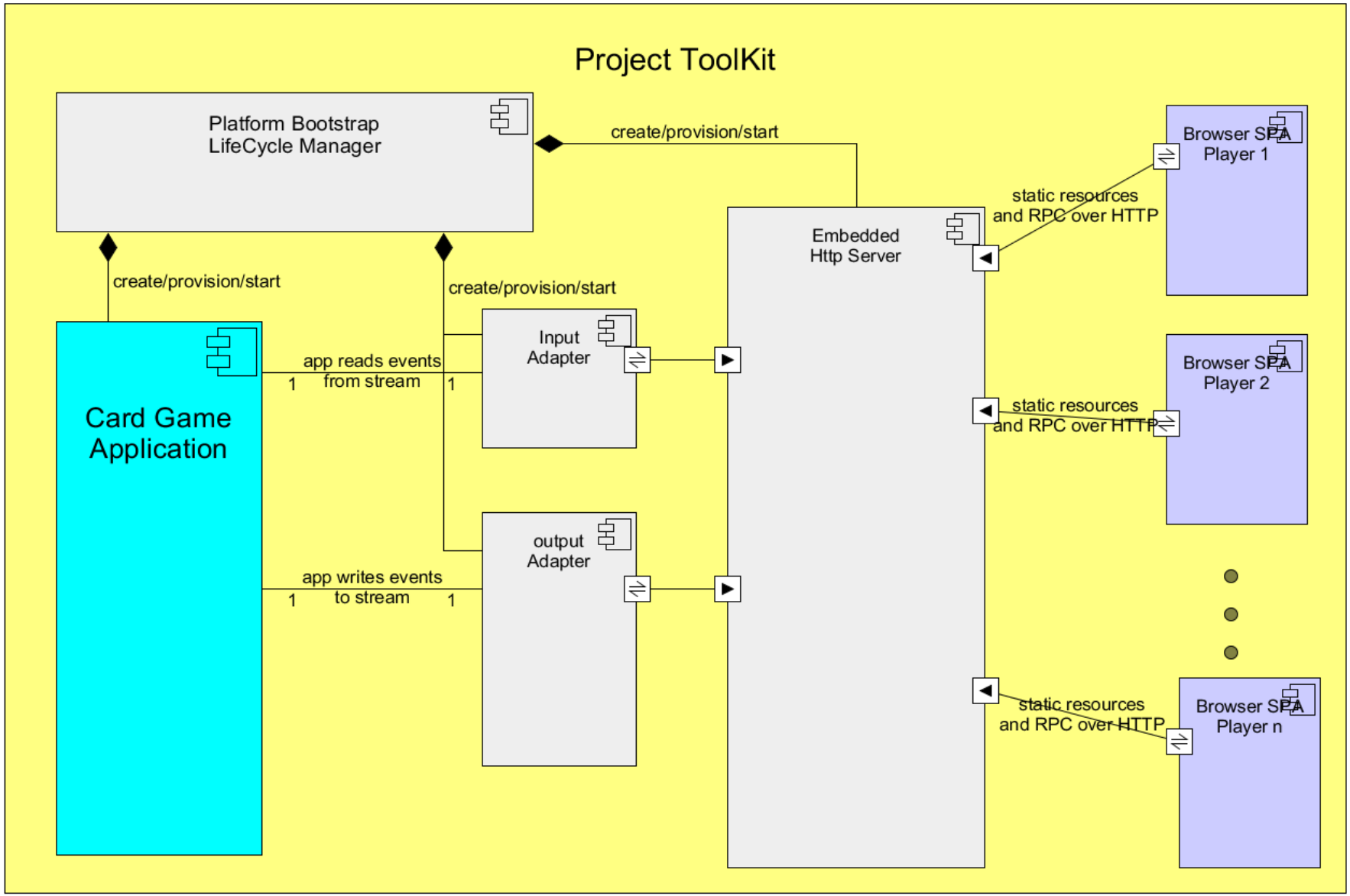
About “Play”

- When does Player input become a move?
  - What if it is a duplicate move?
  - What if it isn’t player’s turn?
  - What if it is an illegal move?
- How do we know what “kind” of move it is?
  - Does it capture?
  - Does it score?
  - Does it end the game?
- The rules examine the input and the state of the game to produce a “move.”

About “Play”

- What do we do with a move?
  - update the state of the game
  - update the presentation
  - switch players? or ask for next phase of player’s current move?
- Thus, a move == a collection of command objects
  - created by the rules
  - applied (by the PlayController) to the table and to the presentation.
  - The move includes different commands for table and presentation.

Platform Architecture



More details

- Note the role of Abstract Factory
- Note which part changes when you add a new game
- Note that those changes are nearly 100% additions of new code, not changes to existing code.

Summary

- This is NOT a complete design.
  - No end of game detection.
    - Can you figure out where to add that to FiftyTwo?
  - No tests or test harness components
  - No abstract factory
  - No inbound event factory
  - Some interfaces still missing
  - Some interfaces may need broadening
  - Lots to resolve and learn.