Design

Phase 3

NOTE 1

- All the way we have been using the OO paradigm
- **All software is not OO**...!
  - 3D graphics is built on an rendering pipeline
  - Relational databases are not OO!
  - The web is not OO
  - ...
- If using any above have to incorporate different paradigms in one application
  - ...hard...
- **Not covered in course (avoid if newbie)!**

Design

- For now the task is to create a runnable version of the domain model i.e. the **design model**
  - Implicit design issue are...
  - ... high **testability**
  - ... implies minimizing dependencie (remove or simplify associations)

There are many levels of design : Design of the model, detailed design of classes (or a few classes), system design (subsystems, GUI,...) ...and more

Dynamic Model

- The domain model is a static view
- As a transition from domain to design model we analyse how to objects in the domain model interact, i.e. create a **dynamic model**
- Visualized as an UML **sequence diagram**
- Possible new objects (not in domain model) will show up
  - Boundary objects, represents the borders of the system (input, output). Example: GUI components
  - Control objects...

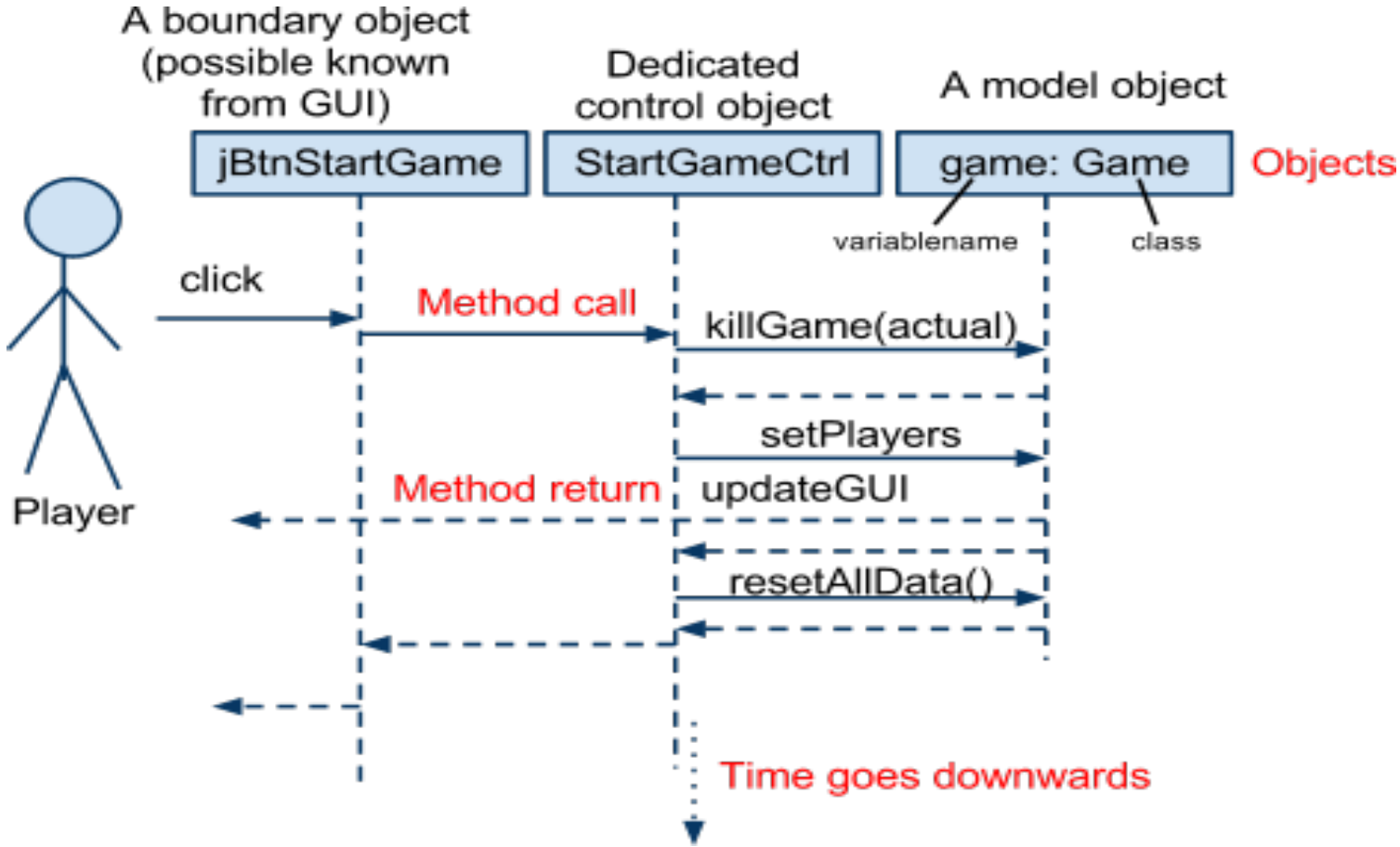
Control Objects

- Control objects uses domain objects to fulfill some functionality (run use case)

- Variations
  - Having "toplevel"-domain objects (i.e. Game, Shop,...) as controls
  - Having dedicated control classes (matching the use cases, DoMoveCtrl)
    - Can extend or include (as in a use case diagram)
  - .. or a mixture of both

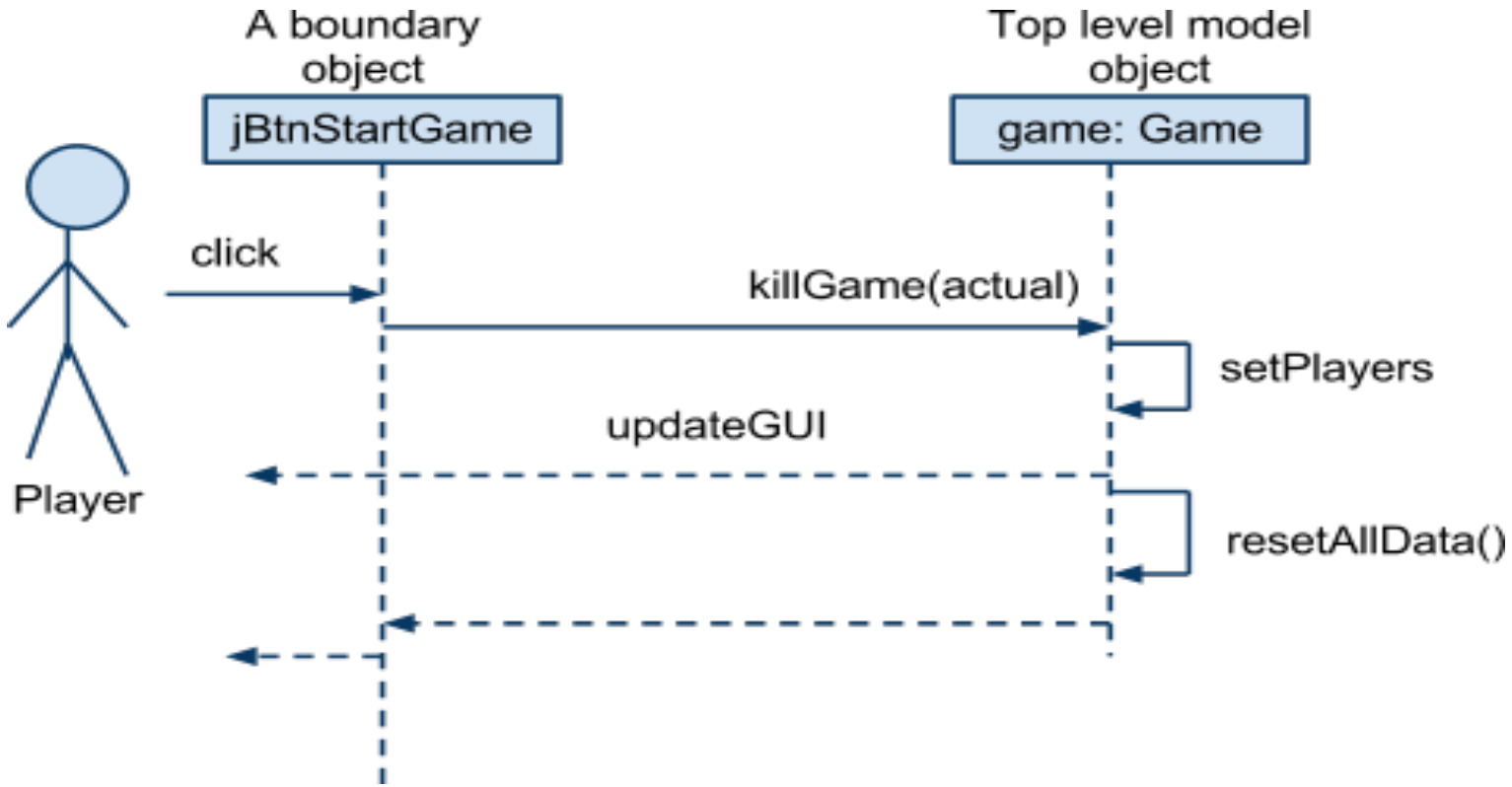
UML for Dynamic Model (1)

- Sequence diagram (dedicated control object)



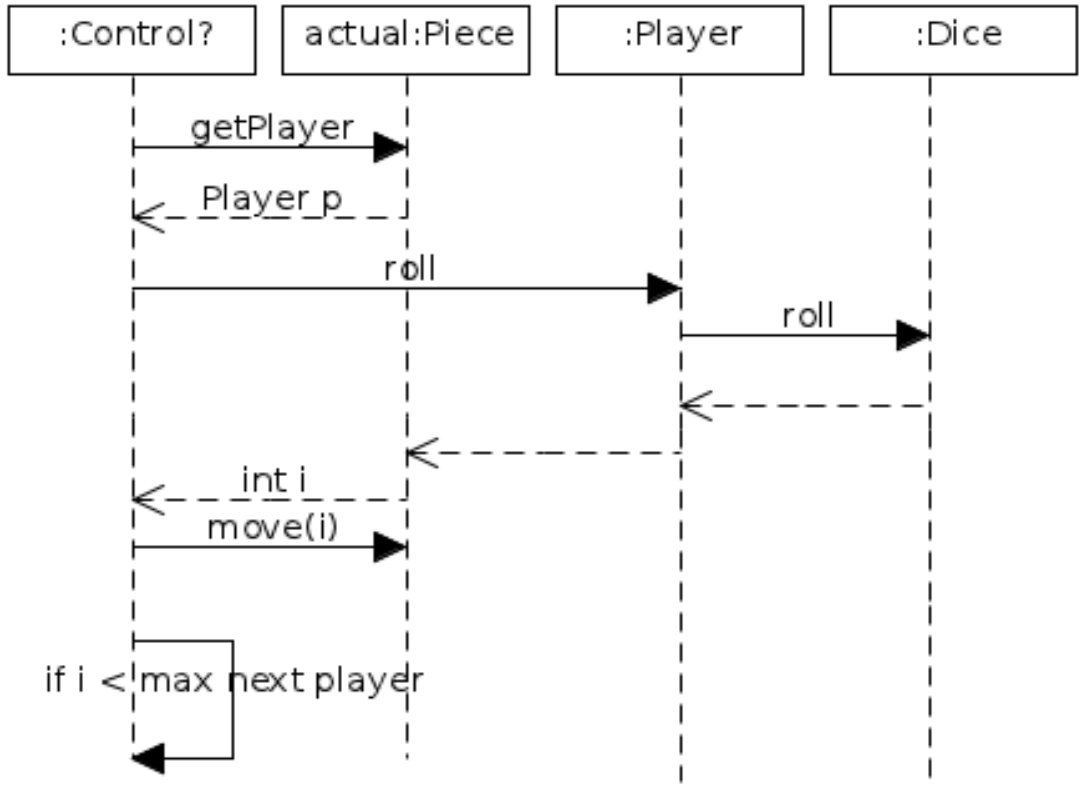
UML for Dynamic Model (2)

- Sequence diagram (top level domain object as control)



Dynamic Model for MP 0.1

UC : Move



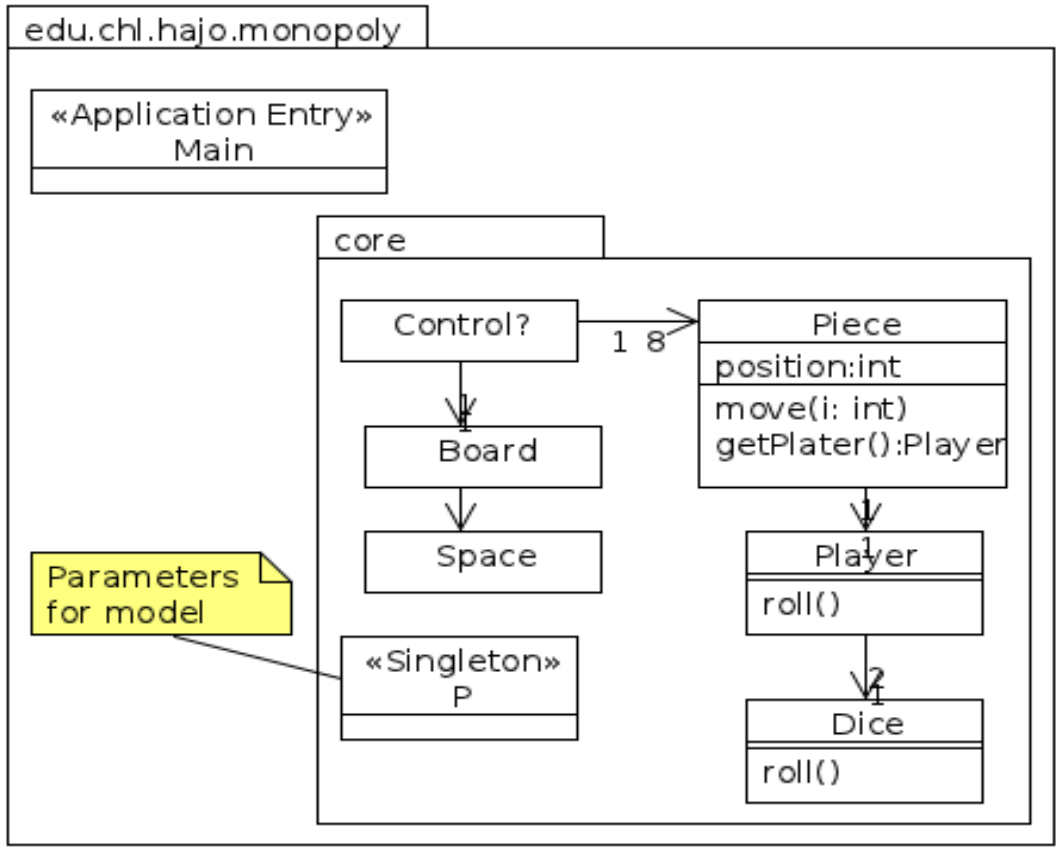
Design thougths: Piece is what we move (not player). Piece must have a position..? Piece knows player but not other way. There's always an actual piece. Everything is done with the actual. Player has a dice?? Where did the board go? What will be the control?

Parameterization of the Model

- The model normally has quit a few parameters (in MP, number of players, how much money for each player, number of spaces of board,...)
- Don't want all that spread all over the model
- As a standard I create a singleton class named "P" to hold all parameters (get-method for each parameter)
  - The bad name is justified by...
    - Laziness, less to type
    - It's not a concept, just a collection of parameters

Design Model for MP 0.1

UC : Move



Design thoughts: We shall use packages (core package holds the model). P class for model parameters, Main for main-method (a standard). A long chain from control to dice.roll().. possible rework? We put an index for every space, needed?

Domain Driven Design

- We don't bother about all the services the model needs to become a fully functional program
- We focus on the model
  - The model is the solution to the problem!
- Other parts (possible developed in parallel)
  - GUI
  - Subsystems
  - ... are **not** part of the solution...!! (technical services)
  - Possible will blur the model, leave out for now...
    - Possible mock-up, hard code

Summary

- We started with the domain model
- Selected a high priority use case
- Created a dynamic model to for the use case, using the domain objects
- Used the dynamic model to create a design model (new classes, simplified associations)

Next: From design model to running implementation