17-723: Designing Large-scale Software Systems

Recitation on Brainstorming Design Alternatives

Tobias Dürschmid

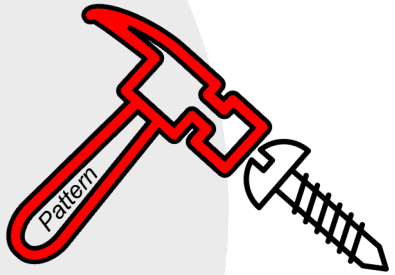
Carnegie Mellon University

Recap of Design Generation Tips

Think of **Many Design Alternatives**

Avoid **Anchoring** to Your Initial Ideas

Start By Considering **Existing Solutions**



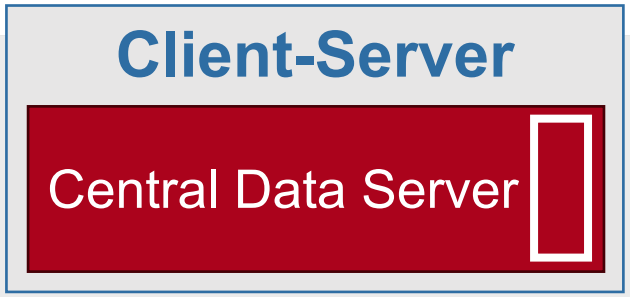
Avoid **Over-Using** Design Patterns

Carnegie Mellon University

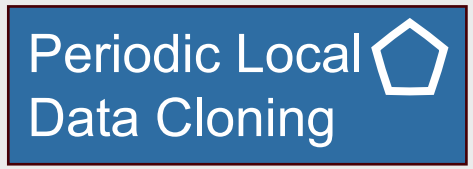
Recap of Brainstorming Steps

1) **Write Ideas on Post-Its**

2) **Cluster Ideas by Similarity**



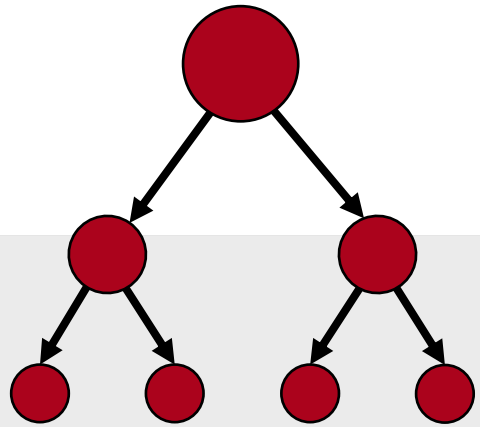
3) **Combine Ideas**



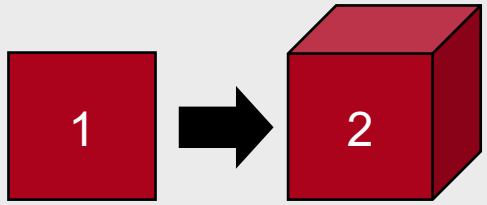
Carnegie Mellon University

Recap of Tips to Solve Complex Design Problems

- **Divide And Conquer To Solve Complex Problems**
  - Split a complex problem into smaller sub-problems



- **Solve Simpler Problems First**
  - Solution to simpler problem might be incomplete, but can be extended later

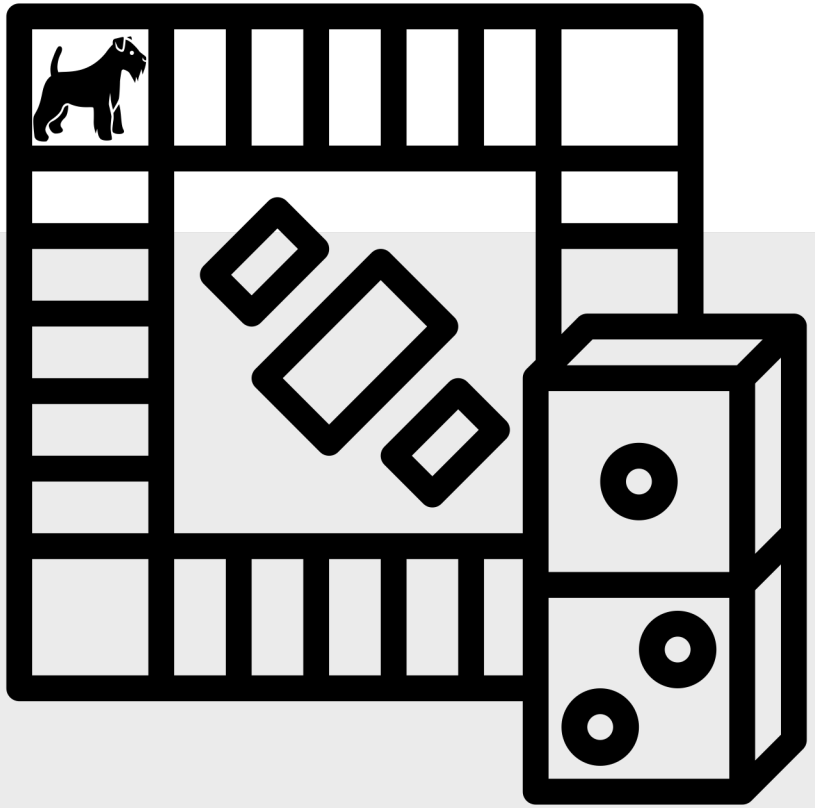


Carnegie Mellon University

Step 0: Divide And Conquer - Identify Sub-Problems

Design a **Digital Monopoly Game!**

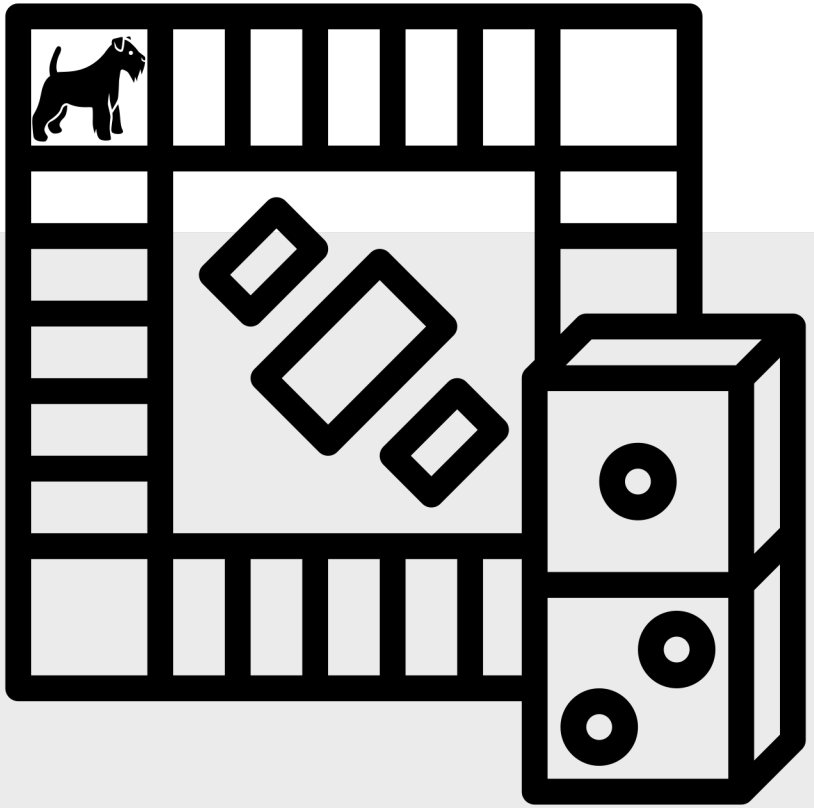
- **Changeability**: The effort to change or replace the UI is minimal
- **Changeability**: The effort to change or replace fields and chance cards is minimal



Carnegie Mellon University

Sub-Problems

- How to **update the UI** (balance, position of players, houses & hotels) when the state of the game changes?
- How to have a diverse range of possible **chance card effects** on the player while allowing **changeability**? (give/charge money, go to jail, go to next railroad, lose houses / hotels, roll again, …)

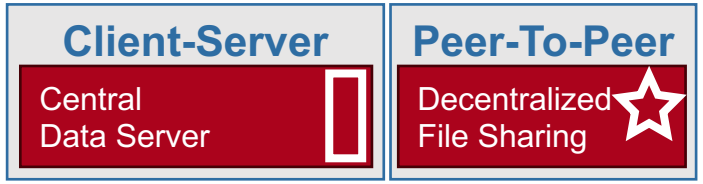


5 min

Step 1: Write Ideas on Post-Its

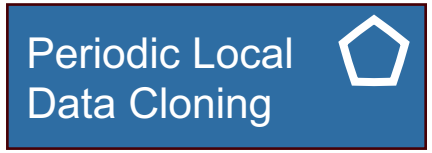
2 min

Step 2: Cluster Ideas by Similarity



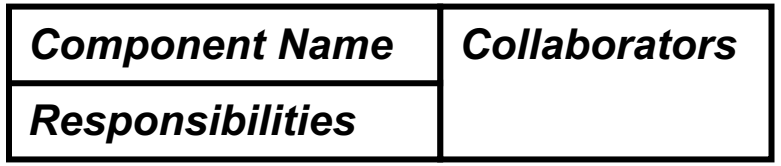
2 min

Step 3: Combine Ideas



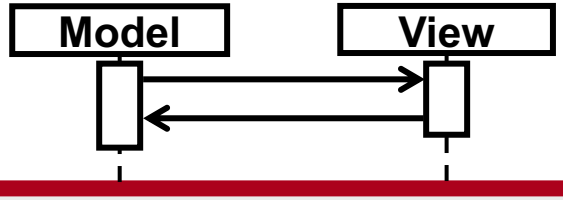
5 min

Step 4: Model CRC Cards



6 min

Step 5: Model Interactions



Carnegie Mellon University

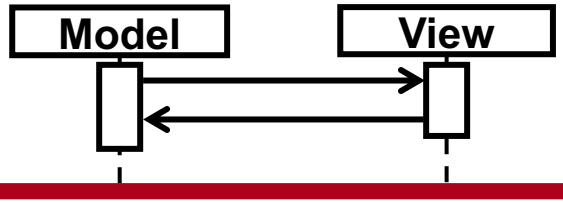
Recap of CRC Cards

A common technique for **modelling software design options**

Class / Component / Role	***Collaborators***
[Name of the component]	[List of other components that this component starts to interact with]
Responsibilities	
[Describe this component's obligations to perform a task or know information]	

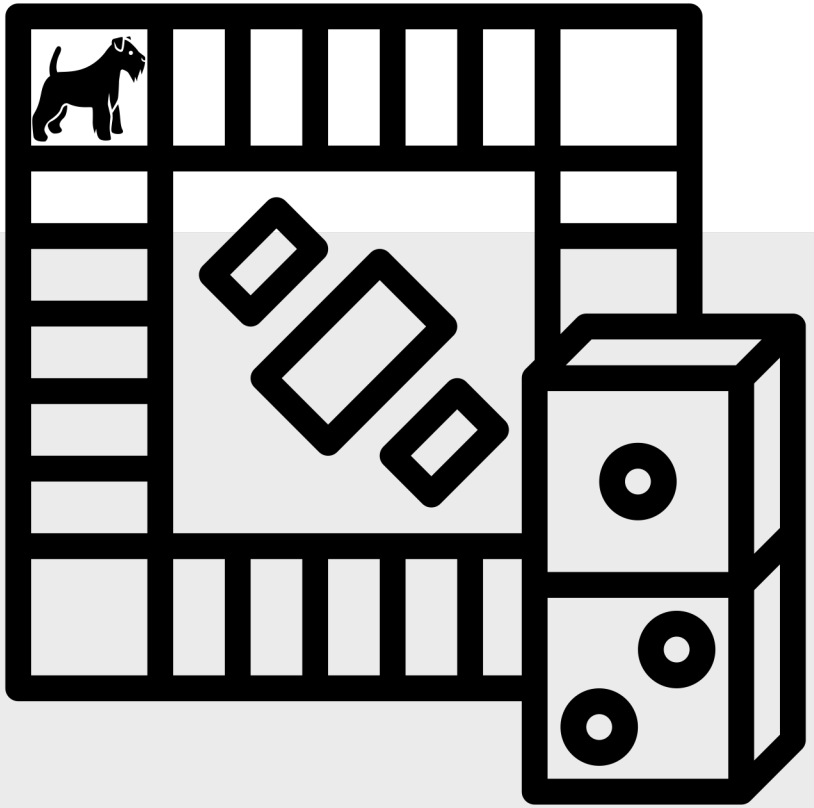
1 min

Step 5: Model Interactions



Generate Design Alternatives!

- How to **update the UI** (balance, position of players, houses & hotels) when the state of the game changes?
- **QA Req** (**Changeability**): The effort to change or replace the UI is minimal
- **Hint**: Consider the sources of updates (what can trigger an update) as well as the UI elements that need to update

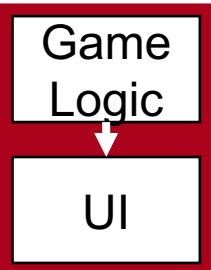
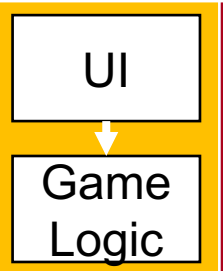


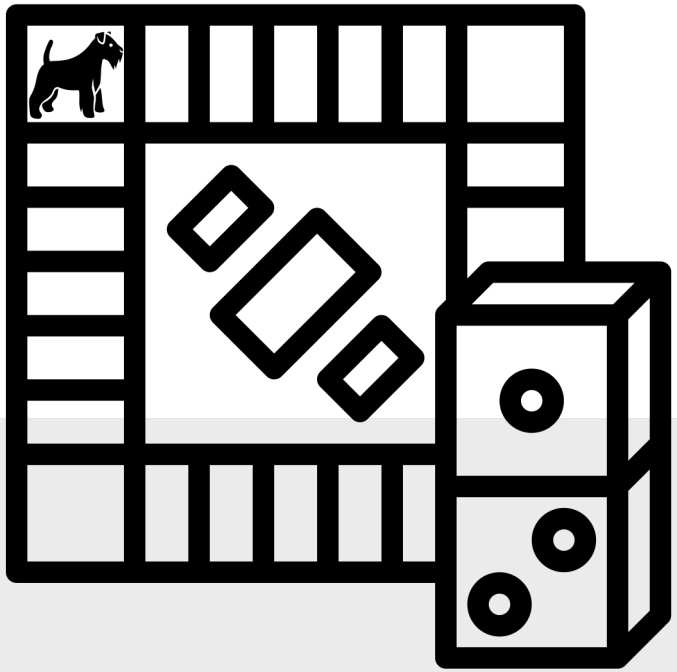
Carnegie Mellon University

See *Façade* Design Pattern

See *Observer* Design Pattern

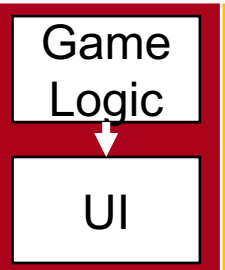
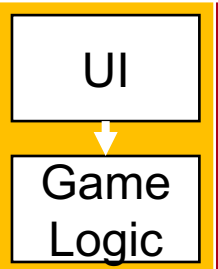
Example UI Design Decision

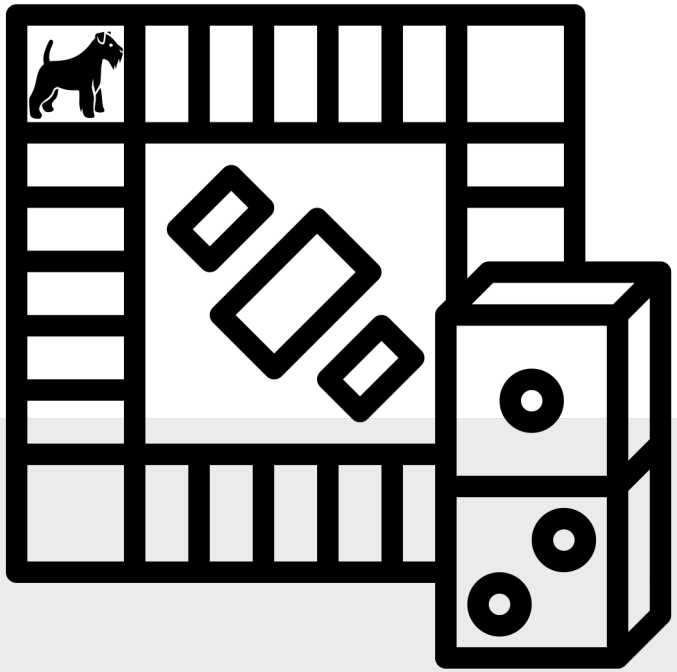
Game component depends on UI 	**UI comp. depends on game component, uses *Observers*** 	**Model-View-Controller**
Changes in game component are passed to a *Façade* of UI-Component. Each object calls UI functions only on the *Façade*. UI input is returned by executing a lambda provided to the *Façade*	*Observers* in UI listen for changes in the models of the game component. UI input is passed to a *Façade* of the game component.	Implement the MVC architectural pattern



Carnegie Mellon University

Example UI Design Decision

Game component depends on UI 	**UI comp. depends on game component, uses *Observers*** 	**Model-View-Controller**
- **Easy** implementation due to simple, direct calls update calls	- **Loose coupling** - Simpler initialization	- **separation of concerns**
- **Initialization** requires a mocked UI -> **complexity**	- Many **small updates**	- Intended for multiple different **simultaneous views** - **Complexity**

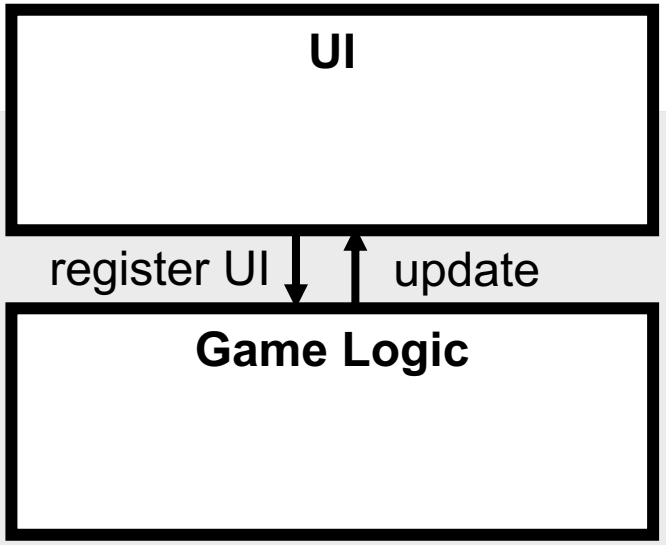


Carnegie Mellon University

Example UI Design Decision

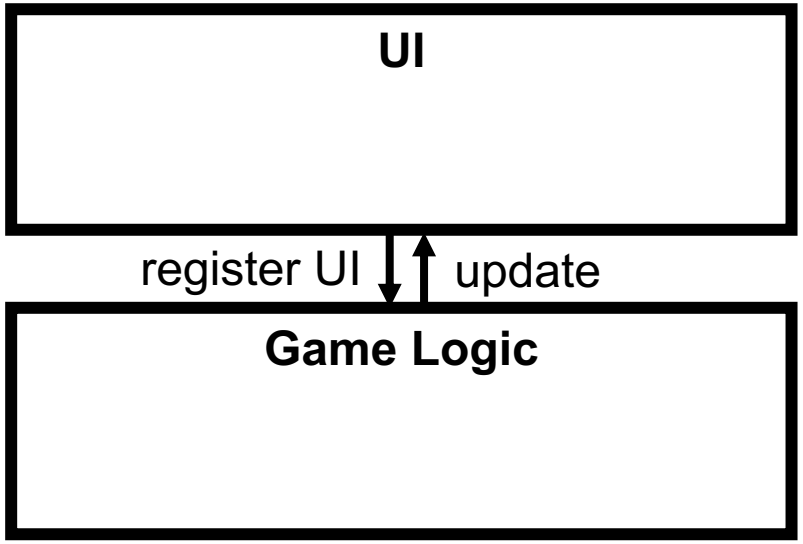
Component: UI	***Collaborators***
Responsibilities Display information Implement update methods Register as observer	Game Logic

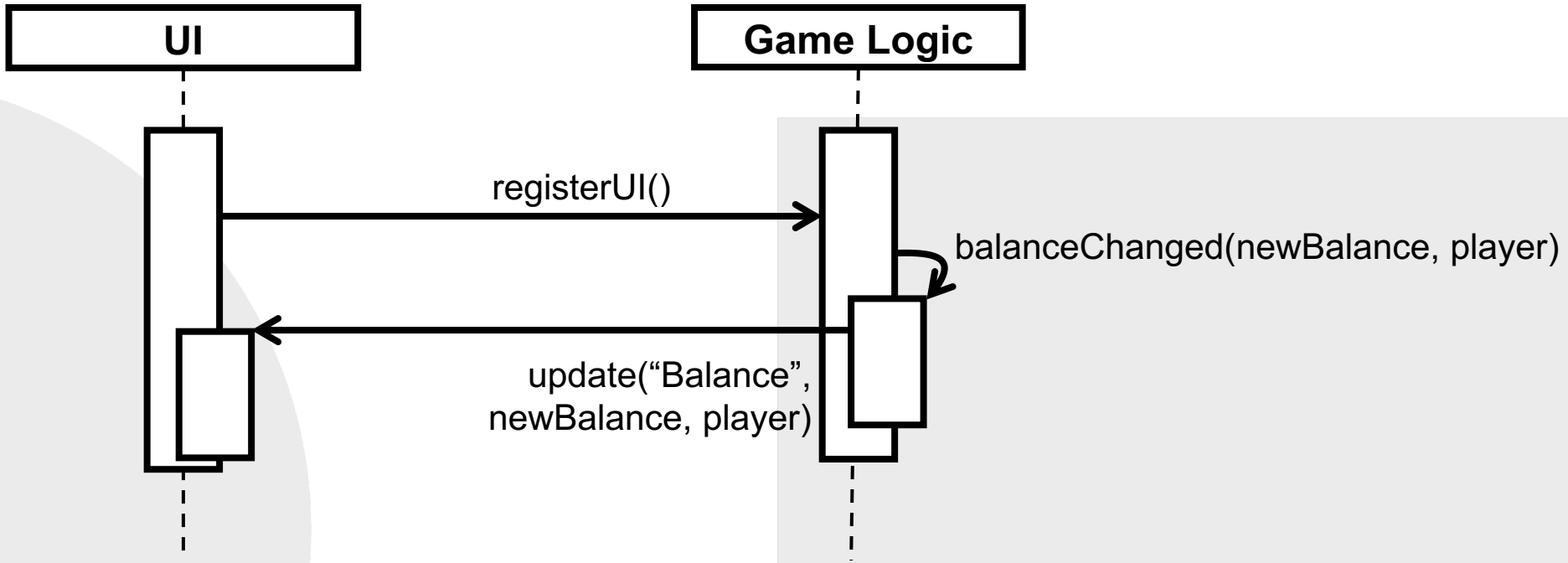
Component: Game Logic	***Collaborators***
Responsibilities Implement the logic of the game Send updates to registered UI	UI



Carnegie Mellon University

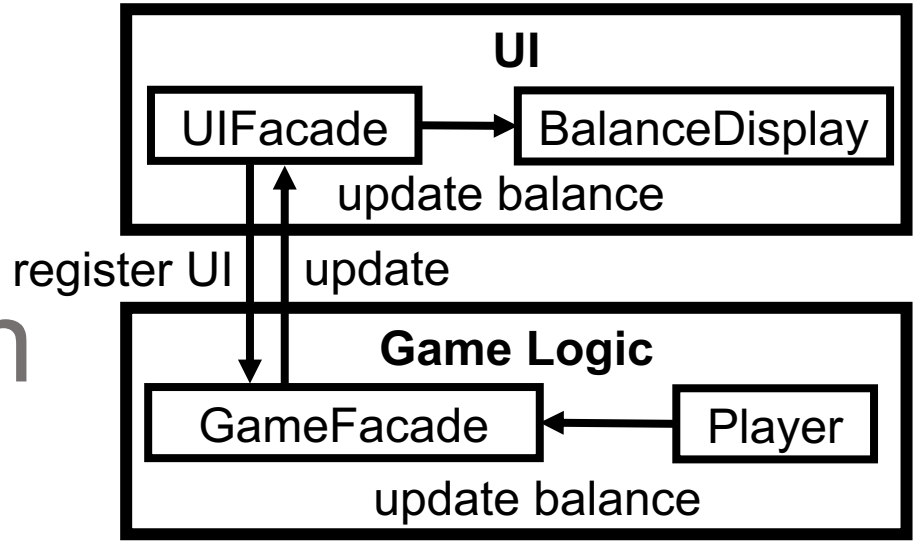
Example Interaction

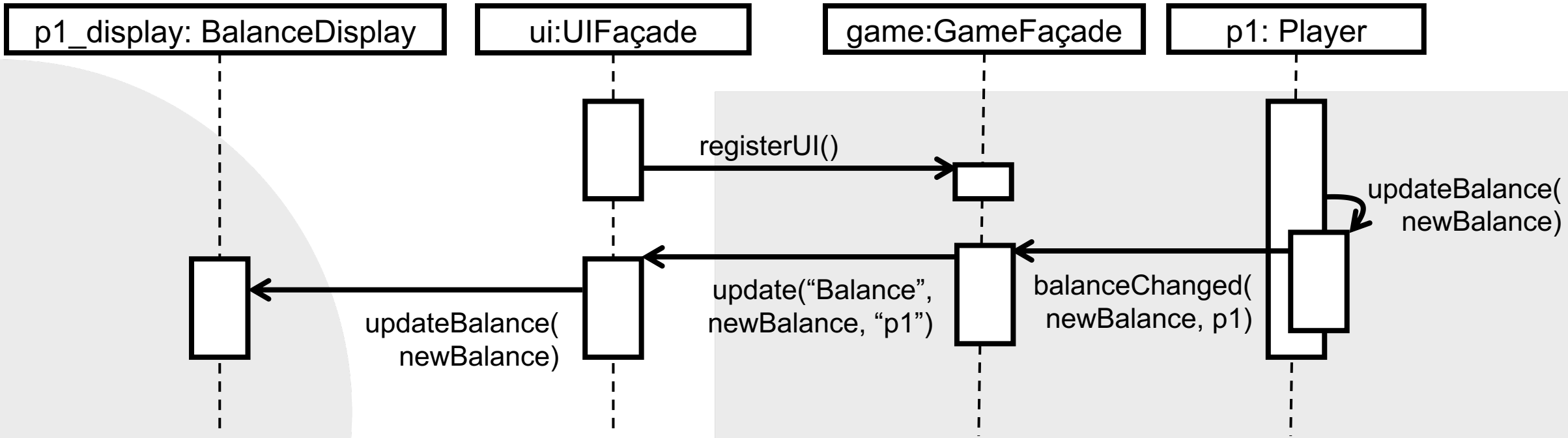




Carnegie Mellon University

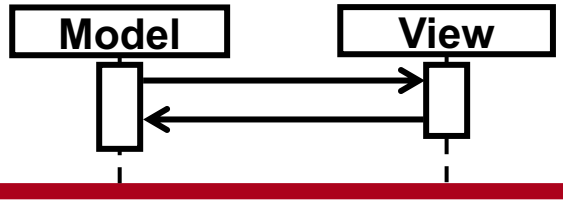
Refined Example Interaction





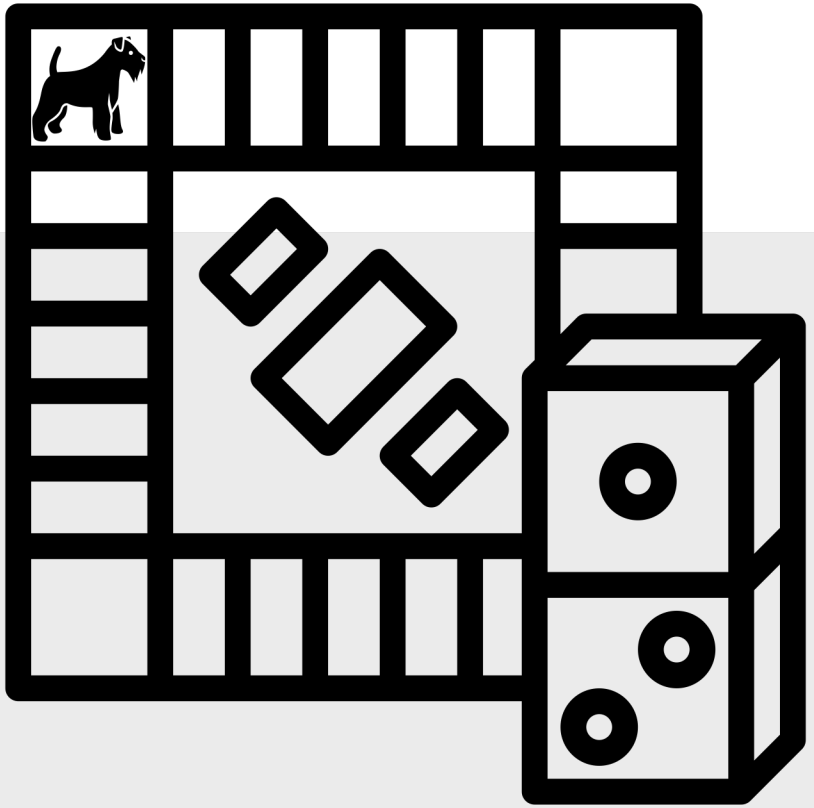
1 min

Step 5: Model Interactions



Generate Design Alternatives!

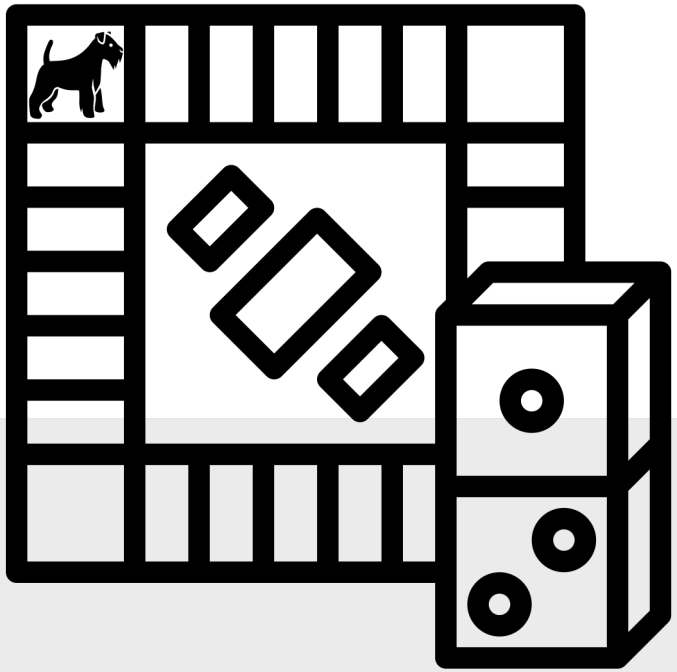
- How to have a diverse range of possible **chance card effects** on the player? (give/charge money, go to jail, go to next railroad, lose houses / hotels, roll again, …)
- **Changeability**: The effort to change or replace fields and chance cards is minimal
- Hint: Also consider how to create and store chance cards.



Carnegie Mellon University

Example Design Decision

Reading from JSON	Abstract Factory Design Pattern
Load cards and fields from a JSON file. Then automatically create an object and call setters using the retrieved data	Implement an Abstract Factory that creates and returns new instances of fields and cards.

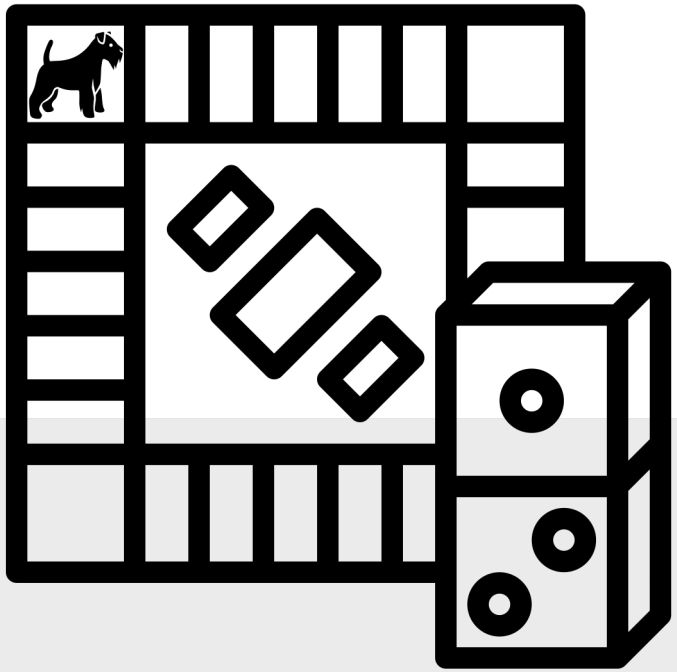


See *Abstract Factory* Design Pattern

Carnegie Mellon University

Example Design Decision

JSON	Abstract Factory
- **Values** can be **changed intuitively** in an external file - **Separation** of code and data	- **Homogenous** with the rest of the software, all written in the same language - **Tools** of the programing environment can be used for manipulation - **Corner fields** can be **identified better** due to method naming
- **Not self-contained** within the software	- Data is **not** easily replaced & modified **with external tools**



Carnegie Mellon University

Summary

- Think of Many Design Alternatives
- Avoid Anchoring to Ideas
- Start By Considering Existing Solutions
- Avoid Over-Using Design Patterns
- Divide And Conquer to Solve Complex Problems
- Solve Simpler Problems First