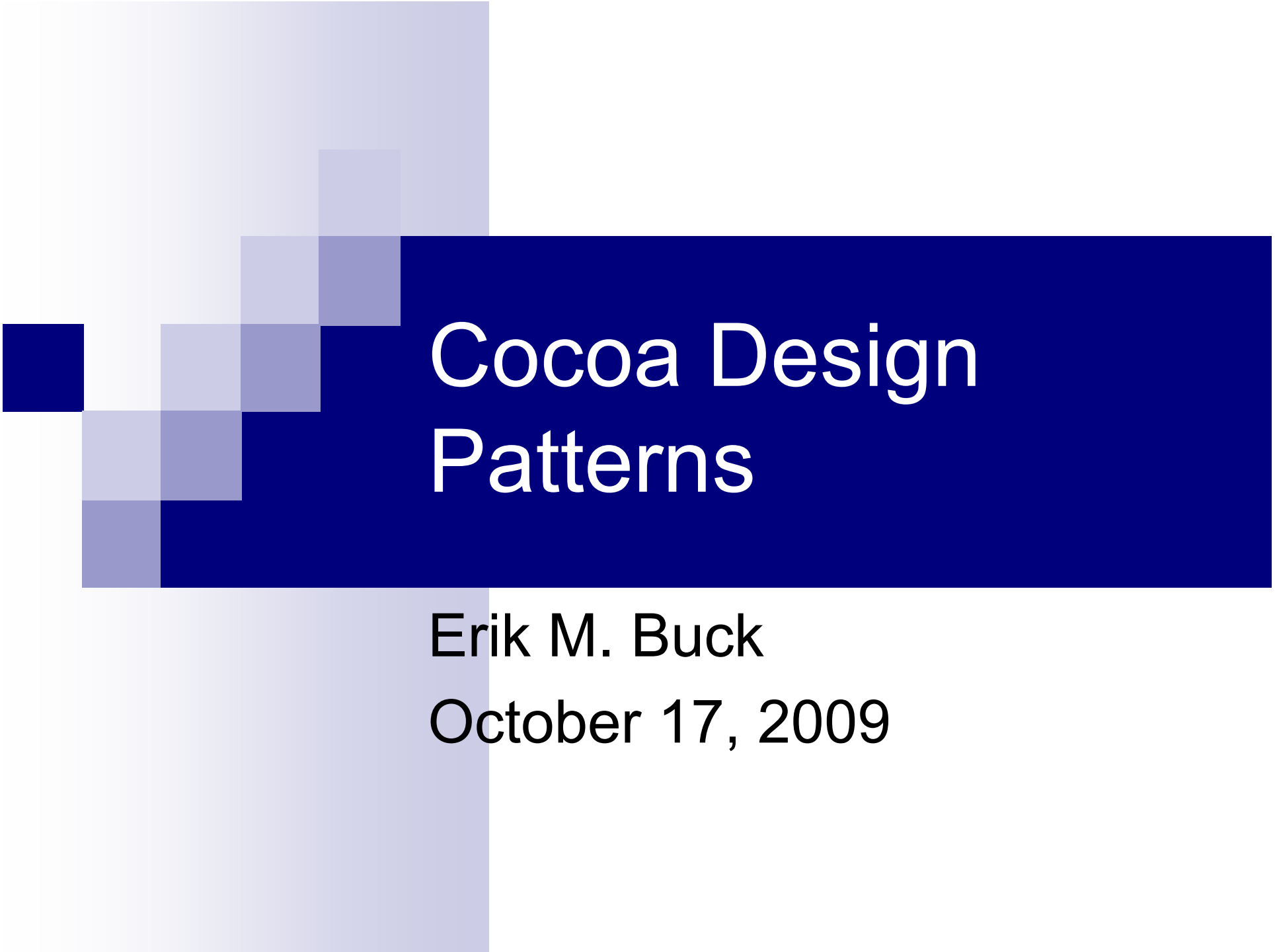
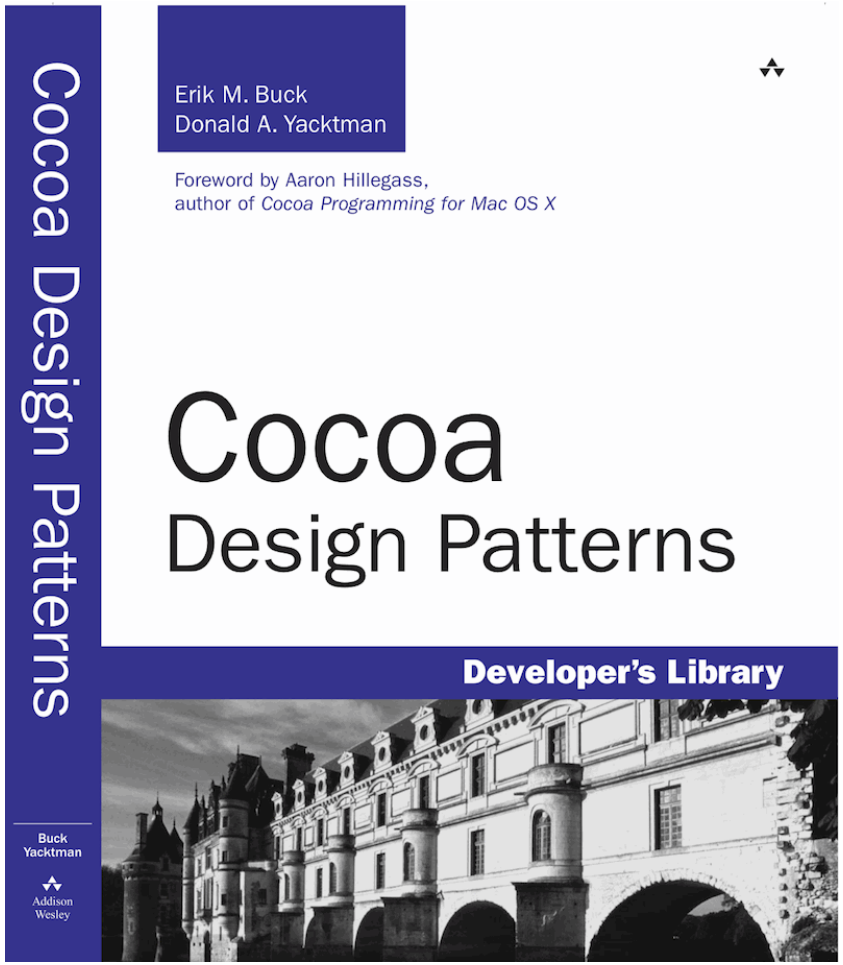


Erik M. Buck

October 17, 2009

Topics



- What is a design pattern?
- Why Focus on Design Patterns?
- What is the Model View Controller design pattern?
- When to use MVC
- When wouldn't you use MVC?
- Pattern implementation example
- Bonus patterns

What is a design pattern?

- Practical solutions to recurring problems
- Don't require amazing programming tricks
- A vocabulary or shorthand to use when explaining complex software
- Not specific algorithms or data structures

What is a design pattern? (continued)

- At a minimum, design patterns contain four essential elements:
  - The pattern name
  - A brief description of the motivation for the pattern or the problem solved by the pattern
  - A detailed description of the pattern and examples
  - The consequences of using the pattern

What is a design pattern? (continued)

- Design Patterns: Elements of Reusable Object-Oriented Software
  - by Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides (1994)
  - ISBN-10: 0201633612
  - ISBN-13: 978-0201633610
- Cocoa Design Patterns by Erik M. Buck and Donald Yacktman
  - ISBN-10: 0321535022
  - ISBN-13: 978-0321535023

What is a design pattern? (continued)

- Web:
  - http://www.cocoadesignpatterns.com
    http://en.wikipedia.org/wiki/Design_pattern_(computer_science)
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html
  - http://c2.com/cgi/wiki?DesignPatternsBook
  - http://www.asp.net/mvc/

Why Focus on Design Patterns?

- Maximize programmer productivity by reducing lifetime software development and maintenance costs
- Object reuse is the answer
- Design patterns identify successful strategies for achieving reuse on a large scale

Guiding Principles of Design

- Solve a problem in a general reusable way
- Ensure that the patterns are flexible and applicable in many contexts
- The same strategies that apply to design of individual objects apply to patterns
- Patterns that involve many objects benefit even more from good object-oriented design than simpler systems

Guiding Principles of Design

(continued)

- Minimize Coupling
  - Coupling refers to dependencies between objects. Dependencies reduce opportunities for reusing the objects independently. Coupling also applies to subsystems within large systems.
  - For example, the Model-View-Controller (MVC) pattern organizes subsystems of classes and is applied to the design of entire applications. The primary intent of the MVC pattern is to partition a complex system of objects into three major subsystems and minimize coupling between the subsystems.

Guiding Principles of Design

(continued)

- Design for Change
  - Designs that are too inflexible ultimately restrict opportunities for reuse. In the worst case, no reuse occurs because it's easier to redesign and re-implement a system than it is to make changes within an existing rigid design.
  - It's possible to anticipate certain types of changes and accommodate them in a design

Guiding Principles of Design

(continued)

- Emphasize Interfaces Rather Than Implementations
  - Interfaces provide a metaphorical contract between an object and the users of the object
  - A contract is necessary for programmers to feel confident reusing framework objects

Guiding Principles of Design

(continued)

- Find the Optimal Granularity
  - Design patterns operate at different levels of granularity. MVC is applied to large subsystems of cooperating classes, but the Singleton pattern makes sure that only one instance of a class is ever created and provides access to that instance.
  - Certain problems are best solved by small patterns that involve only a few classes while other problems are solved by reusing grand overarching patterns.

Guiding Principles of Design

(continued)

- Use Composition in Preference to Inheritance
  - It can't be said enough times that coupling is the enemy.
  - It is ironic that inheritance is simultaneously one of the most powerful tools in object-oriented programming and one of the leading causes of coupling.

What is MVC?

- MVC is a pattern for designing applications that store information, retrieve information, present information to a user, and/or enable a user to edit or otherwise manipulate the information.
- MVC is one of the oldest and most successfully reused software design patterns
- It's a high level pattern for organizing large groups of cooperating objects into distinct subsystems: the Model, the View, and the Controller.

What is MVC? (continued)

- Model

  provides the unique capabilities and information storage for an application.

- View

  Enables user interaction with information gathered from the Model

- Controller

  decouples the Model from the Views

What is MVC? (continued)

- The primary purpose of MVC is to decouple the Model subsystem from the Views so that each can change independently.
- The Controller subsystem provides that decoupling.
- The Controller subsystem is often tricky to design and may seem like it adds needless complexity

What is MVC? (continued)

- The flow of information is between the Model and the Views, so why introduce another layer?
  - The Controller subsystem provides insulation between the Model and the Views
    - Views tend to change much more often than Models
    - There potentially many Views
    - Change the Model without affecting all of the Views

What is MVC? (continued)

- It's usually easier to test a Model directly rather than through a user interface.
  - When testing through a user interface, extra effort is required to determine whether a test failure is the result of a bug in the Model or a bug in the View or both.
- The Model is often developed by one team and the user interfaces are developed by another
  - The skills needed to develop the Model may be very different than those needed to produce an excellent user experience.

When to use MVC

- Editor or viewer applications

  Applications that are editors or viewers for underlying information are naturally organized with the MVC pattern.

  - When there are multiple ways to view the same information
  - When there is a consistent way of viewing different types of information
  - To simplify simultaneous development of Models and Views by teams.

When wouldn’t you use MVC?

- Operating system device drivers

  The application used to configure device drivers might adopt the MVC design, but the drivers themselves must conform to operating system interfaces and don’t usually present information directly to users.

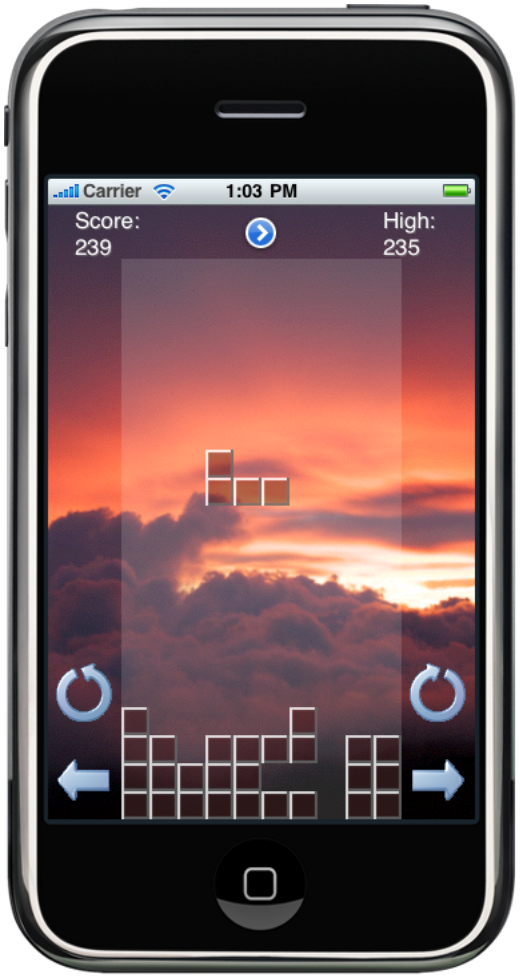
- Long running computation intensive programs.

  Long running calculations are often batch processed and again don’t directly provide a user interface.

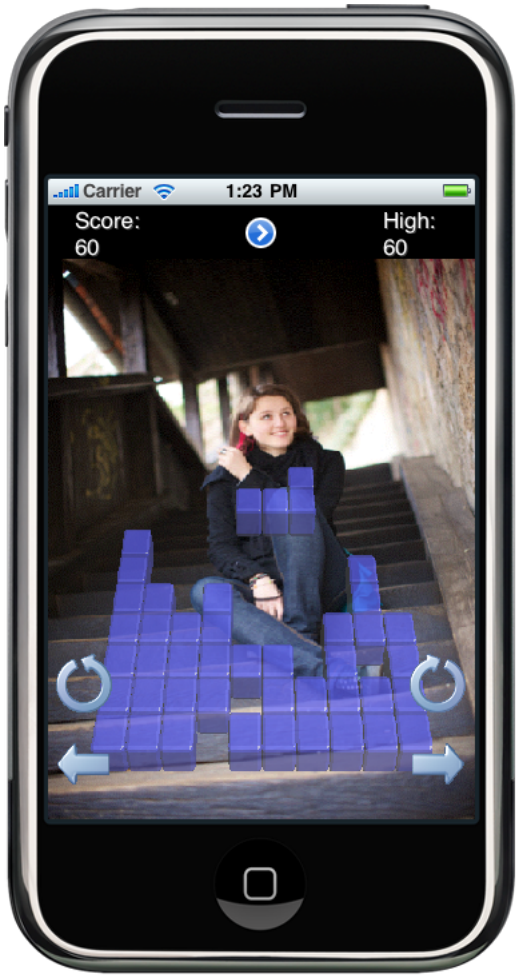
Pattern implementation example

MVC Game

Games present information and enable user interaction

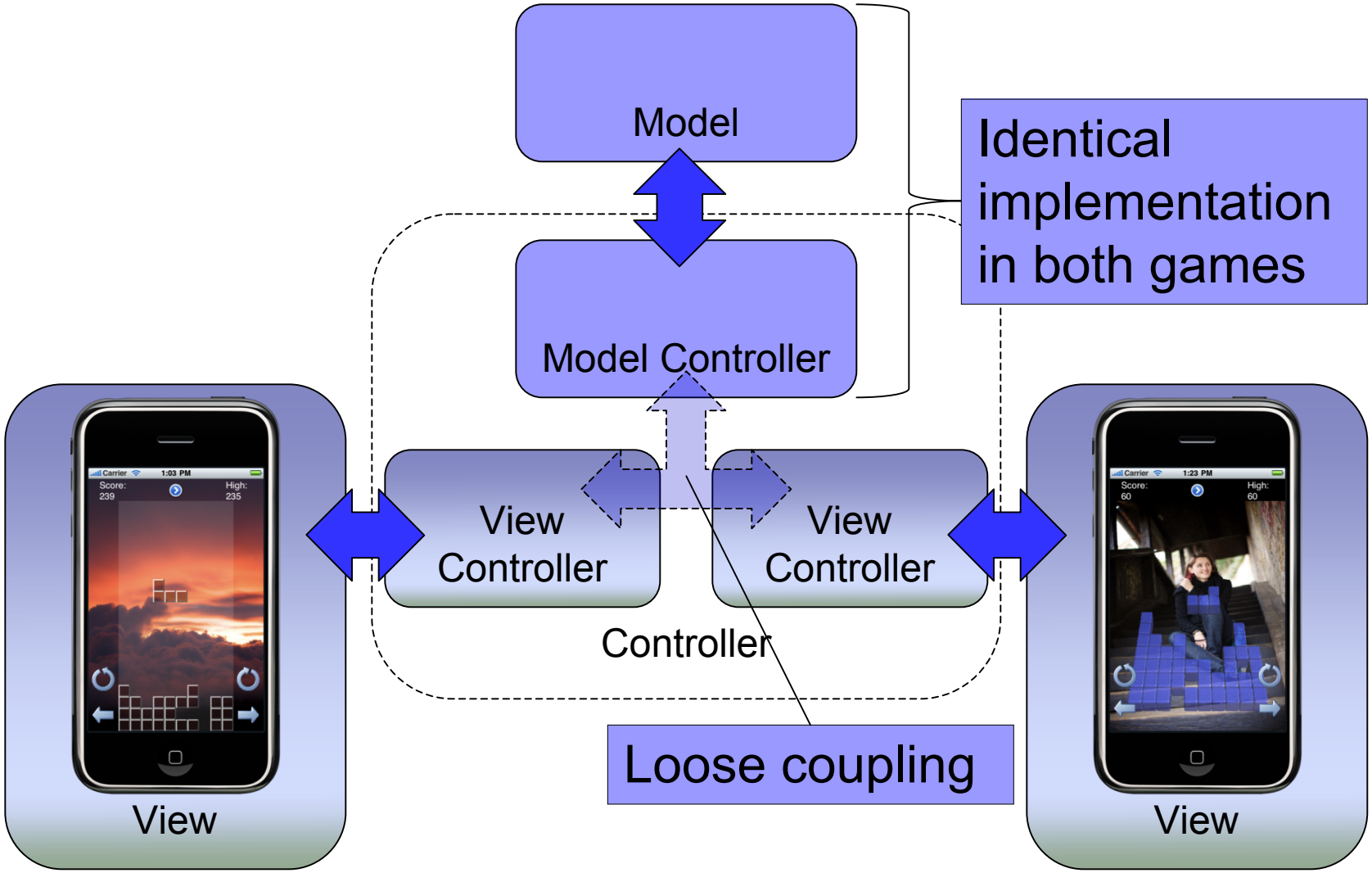


Transparis

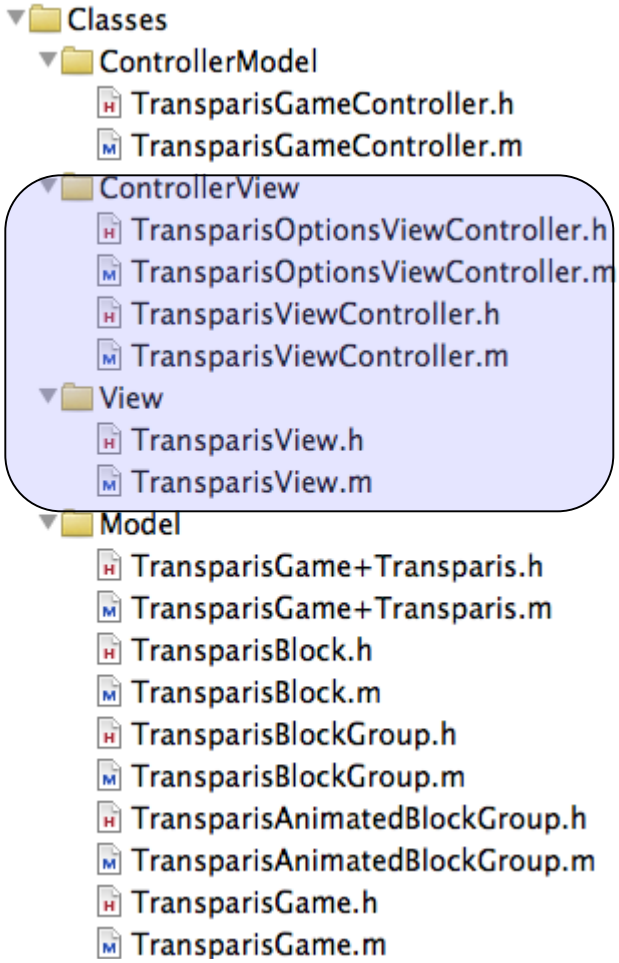


Transparis 3D

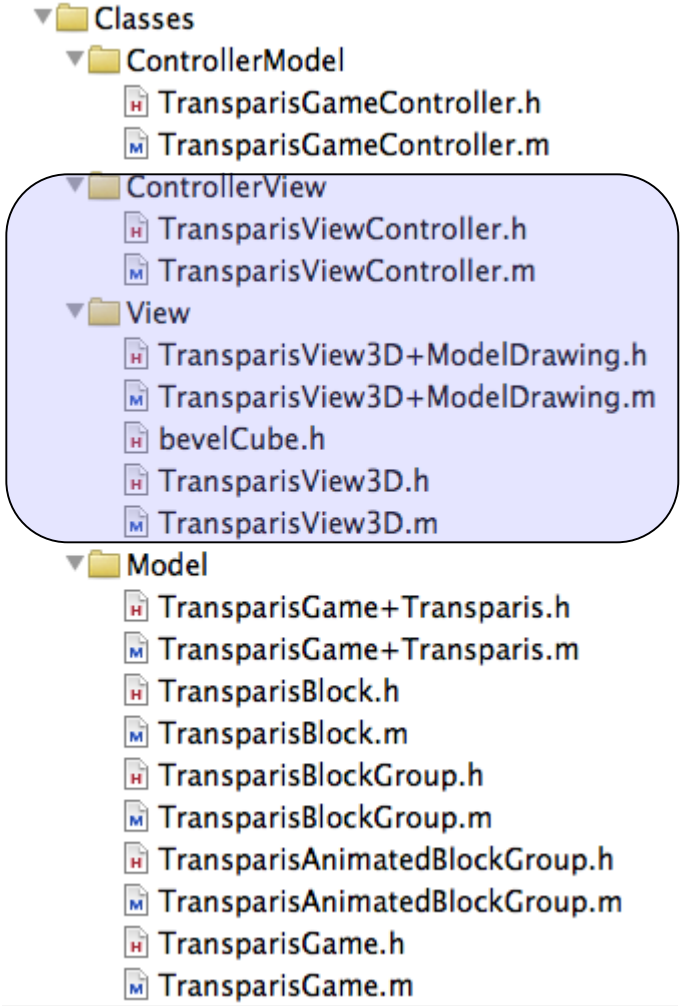
One Model multiple Views



Classes

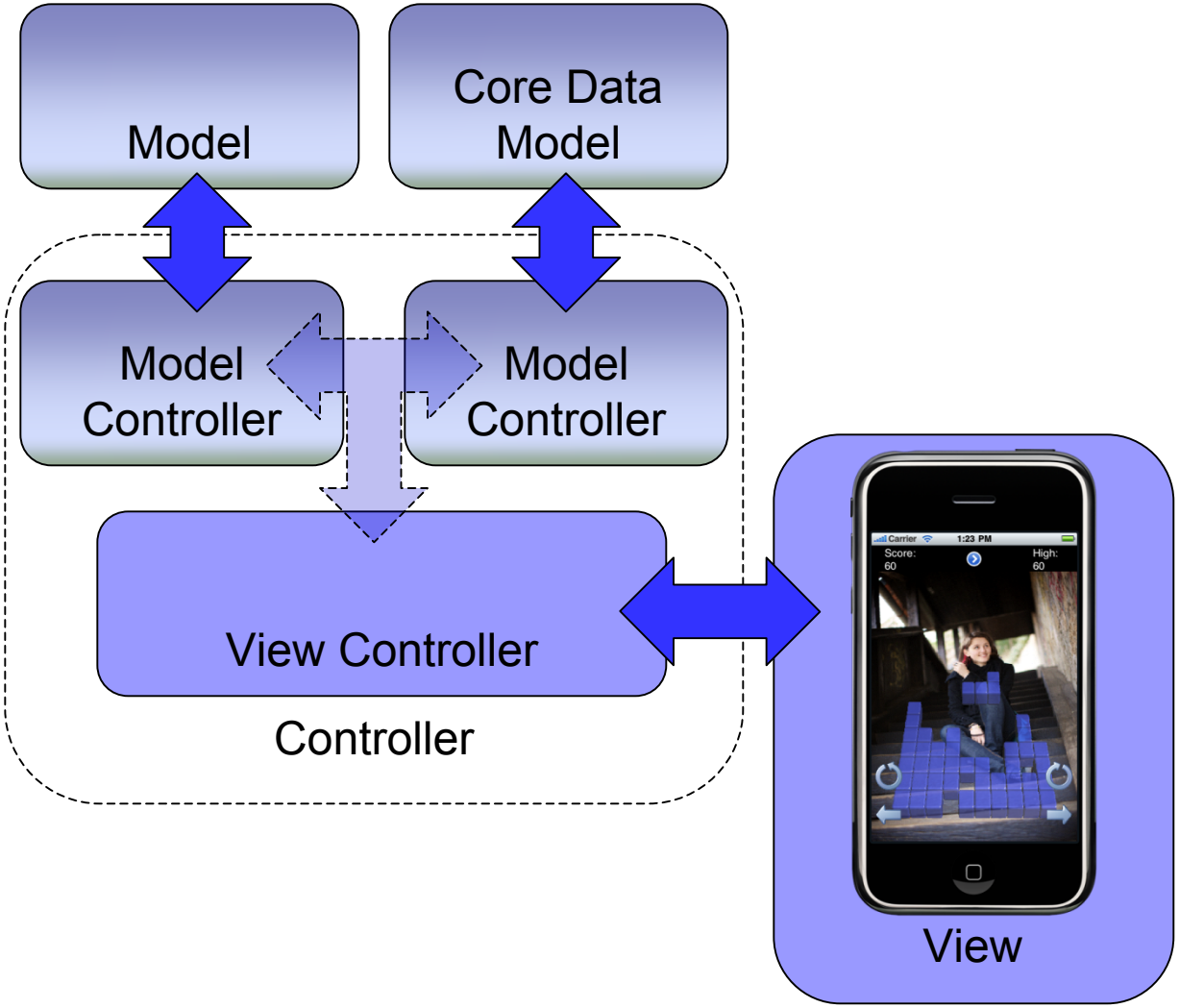


Transparis

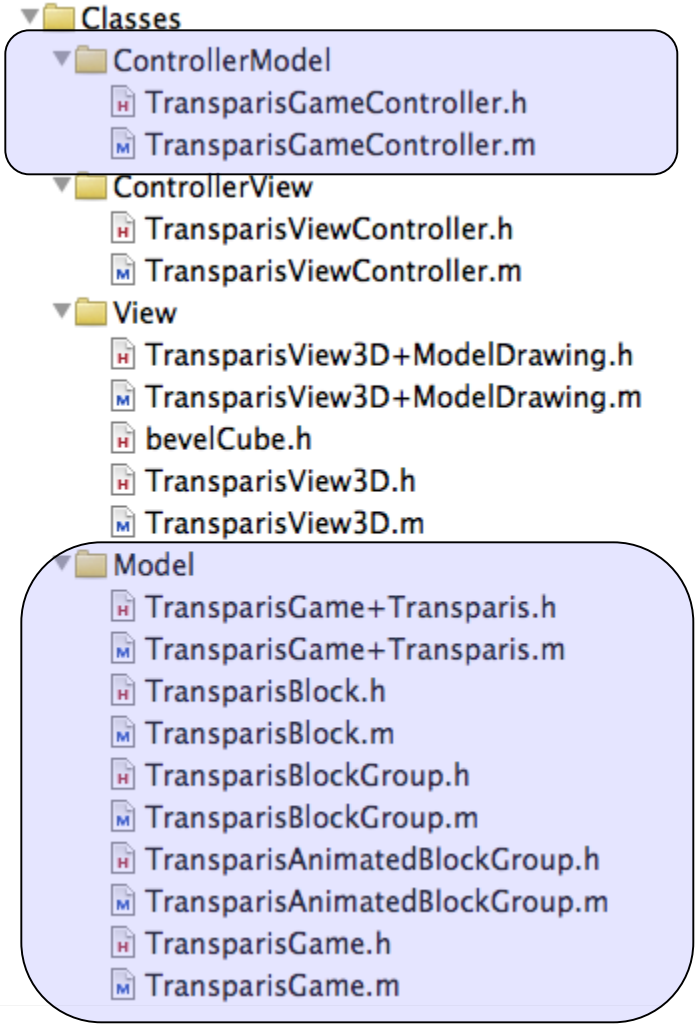


Transparis 3D

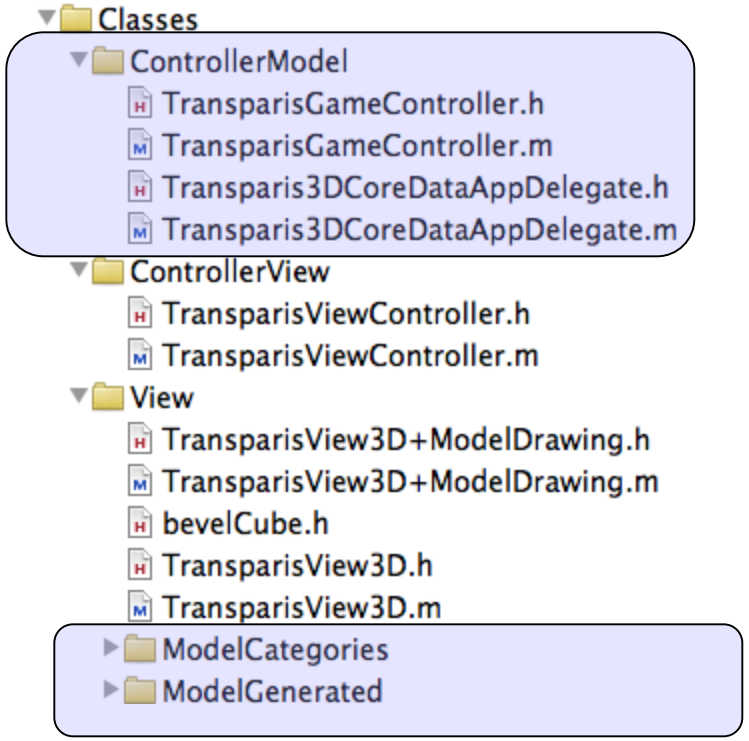
Multiple Models one View



Classes

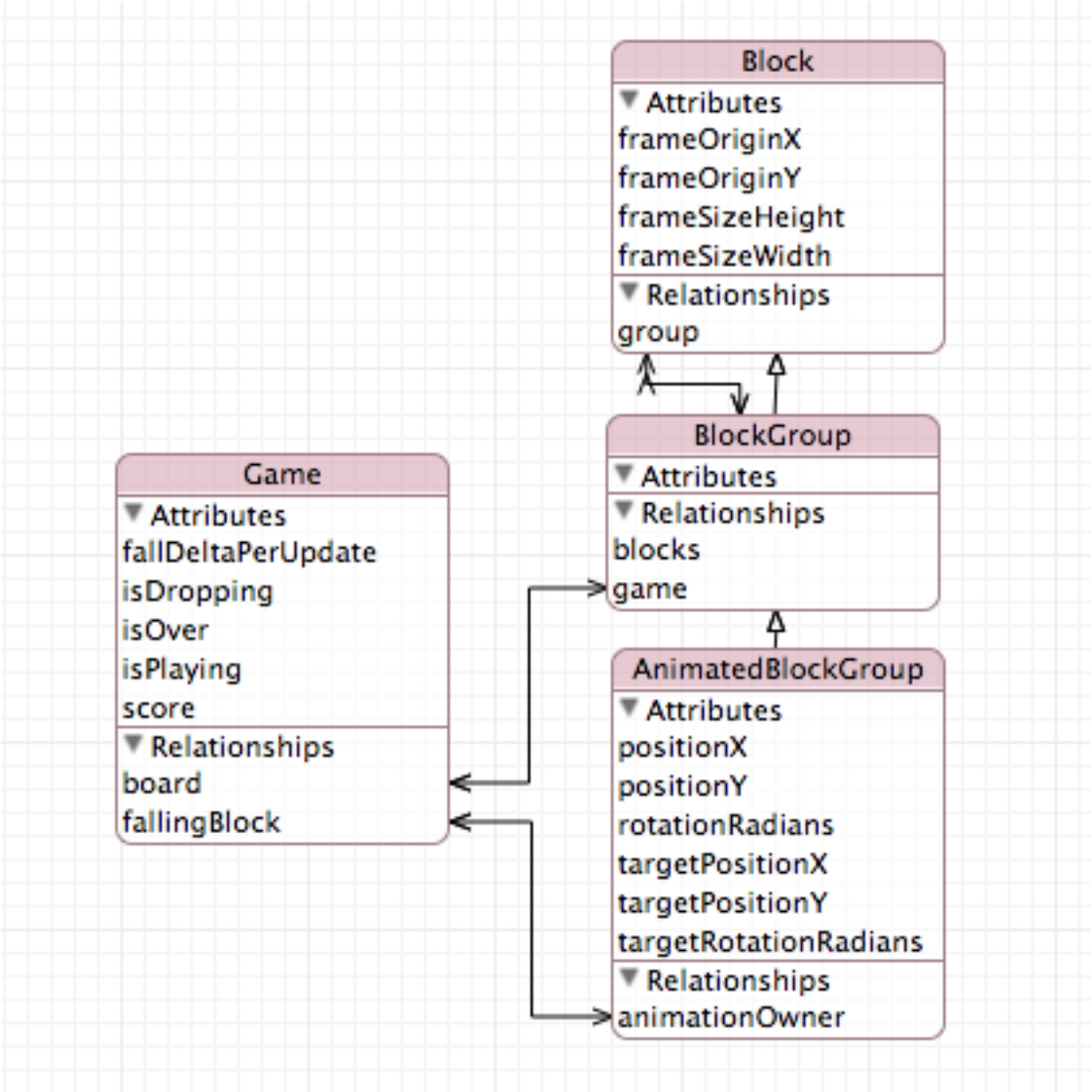


Transparis 3D



Transparis 3D
Core Data

Core Data Model

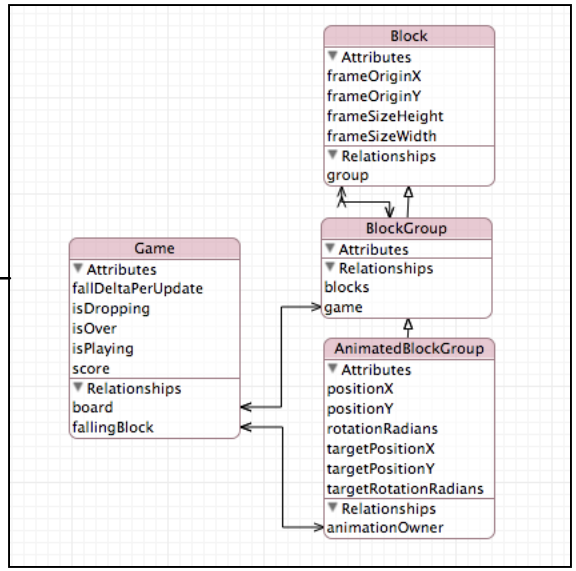


Bonus Patterns

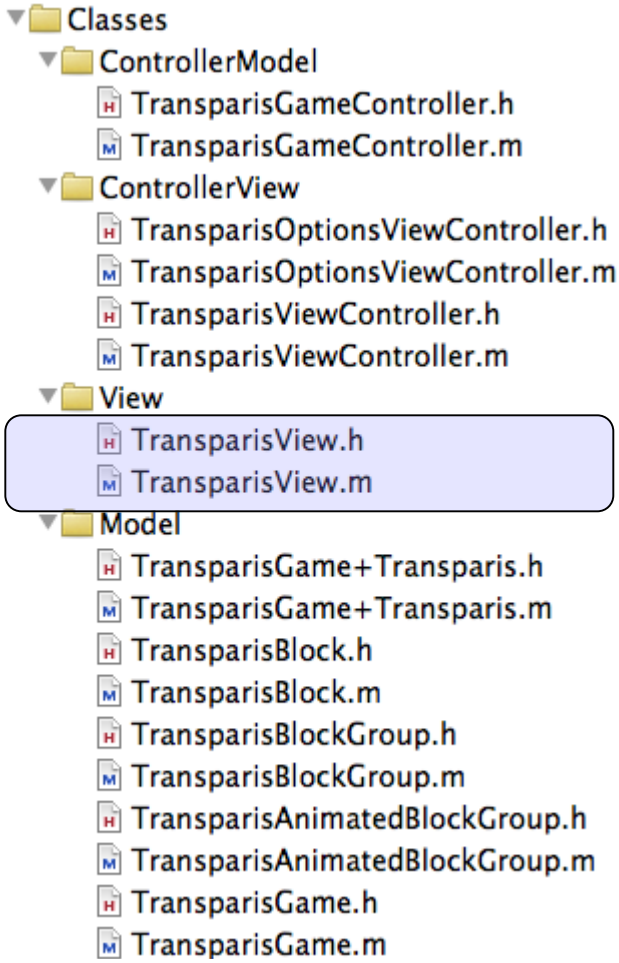
- Category
- Notification

Category: Add methods to existing classes

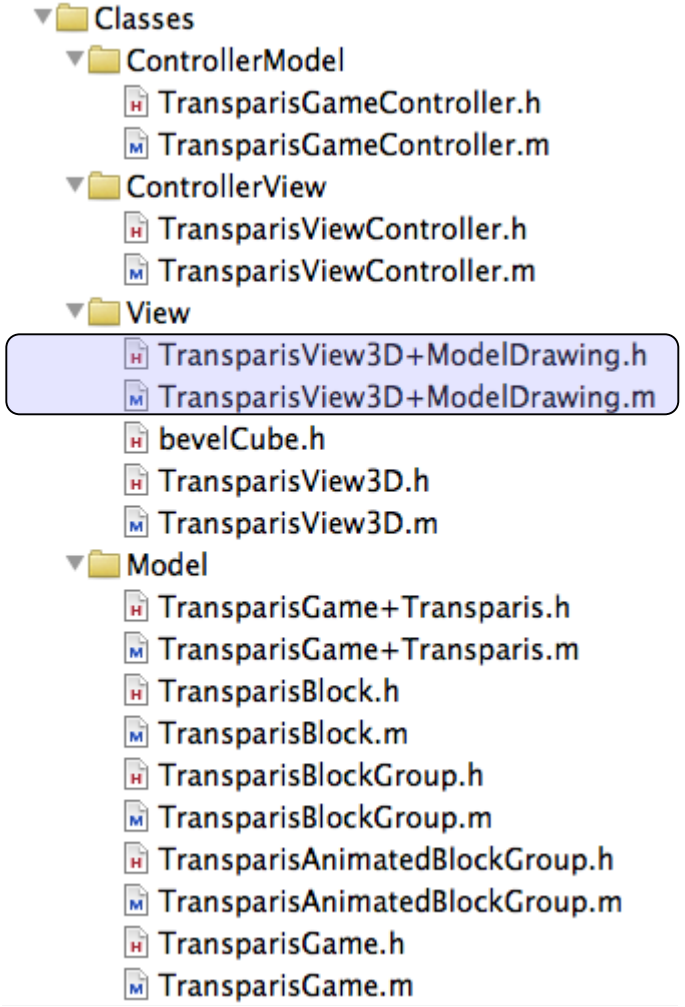
- Classes
  - ModelCategories
    - TransparisAnimatedBlockGroup+Transparis.h
    - TransparisAnimatedBlockGroup+Transparis.m
    - TransparisBlock+Transparis.h
    - TransparisBlock+Transparis.m
    - TransparisBlockGroup+Transparis.h
    - TransparisBlockGroup+Transparis.m
    - TransparisGame+Transparis.h
    - TransparisGame+Transparis.m
  - ModelGenerated
    - TransparisGame.h
    - TransparisGame.m
    - TransparisBlockGroup.h
    - TransparisBlockGroup.m
    - TransparisBlock.h
    - TransparisBlock.m
    - TransparisAnimatedBlockGroup.h
    - TransparisAnimatedBlockGroup.m



Category: Implementation locality



Transparis



Transparis 3D

Notification: Communicate without coupling