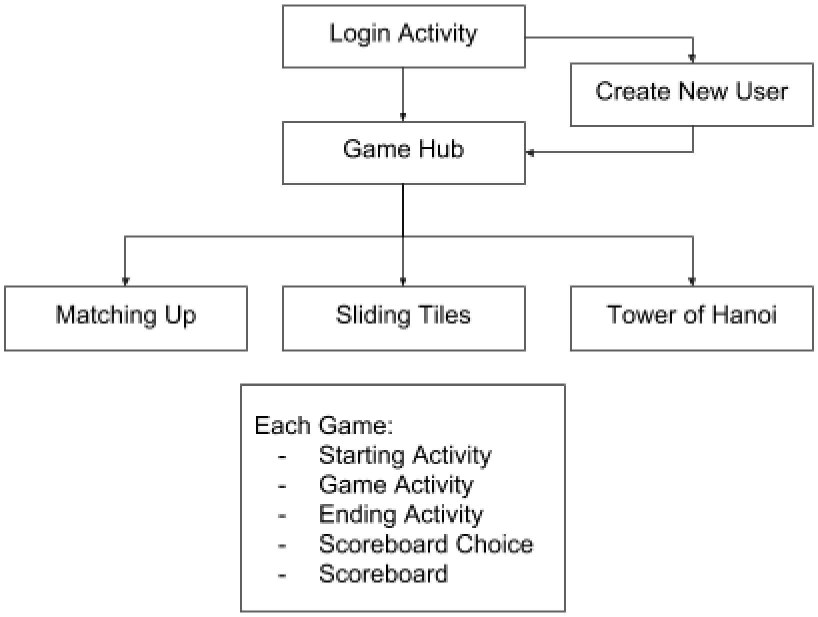
WALKTHROUGH

group_0650: hungch11, shinjisu, zhangg37, thomasp6, hongkar1

Tuesday December 4th 2018 3-3:30 pm

Overview of Pages in Application



All Classes

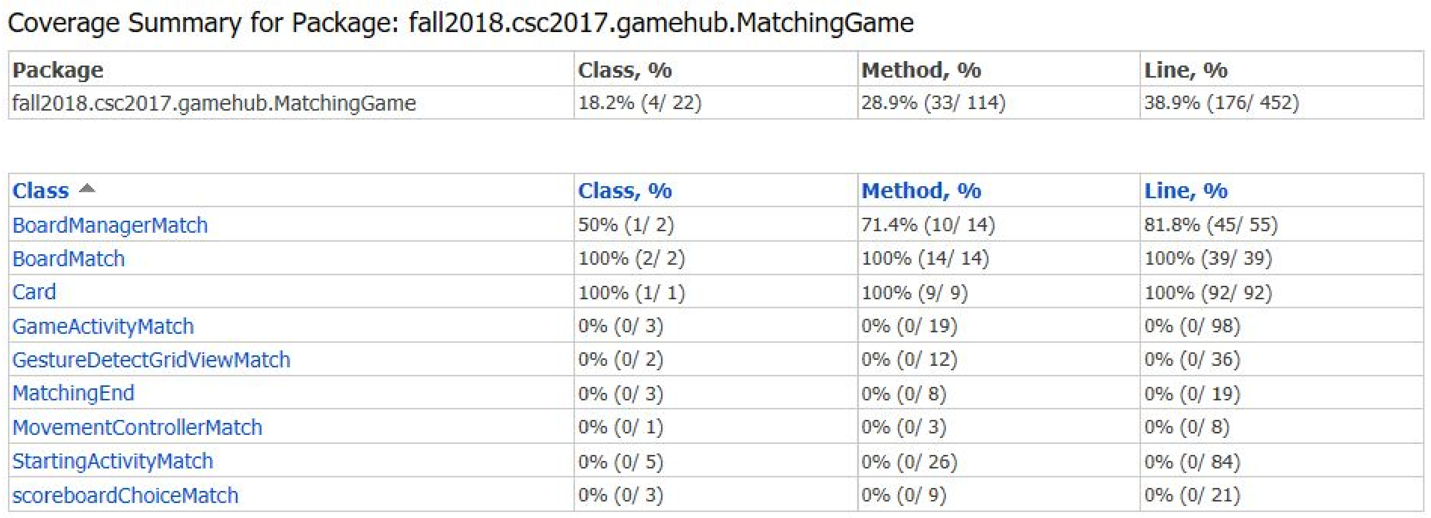
Classes covered by unit tests are highlighted in green

Classes Accessed Throughout App in the Activity Classes ● UserManager ● User ● ScoreboardManager ● ScoreBoard ● DataTuple		**Classes only in Login & Game Hub** ● LogInActivity ● CreateUser ● GameHub
Classes used for Match Up ● StartingActivityMatch ● GameActivity ● MatchingEnd ● BoardManagerMatch ● BoardMatch ● Card ● CustomAdaptor ● GestureDetectGridViewMatch ● MovementControllerMatch ● ScoreboardChoiceMatch ● ShowScoreboard ● ScoreListAdapter	**Classes used for Sliding Tiles** ● StartingActivity ● GameActivity ● SlidingTileEnd ● BoardManager ● Board ● Tile ● CustomAdaptor ● GestureDetectGridView ● MovementController ● ShowScoreboardChoice ● ShowScoreboard ● ScoreListAdapter	**Classes used for Tower of Hanoi** ● StartingActivity ● GameActivity ● HanoEnd ● TowerOfHanoManager ● DrawBricks ● DrawRingsAndBackground ● ShowScoreboardChoiceHano ● ShowScoreboardHano ● ScoreListAdapter

UNIT TEST COVERAGE

Note: Classes with 0% coverage have methods that are all android associated so they aren't tested as they require a mock or instrumental testing.

MatchingGameTest



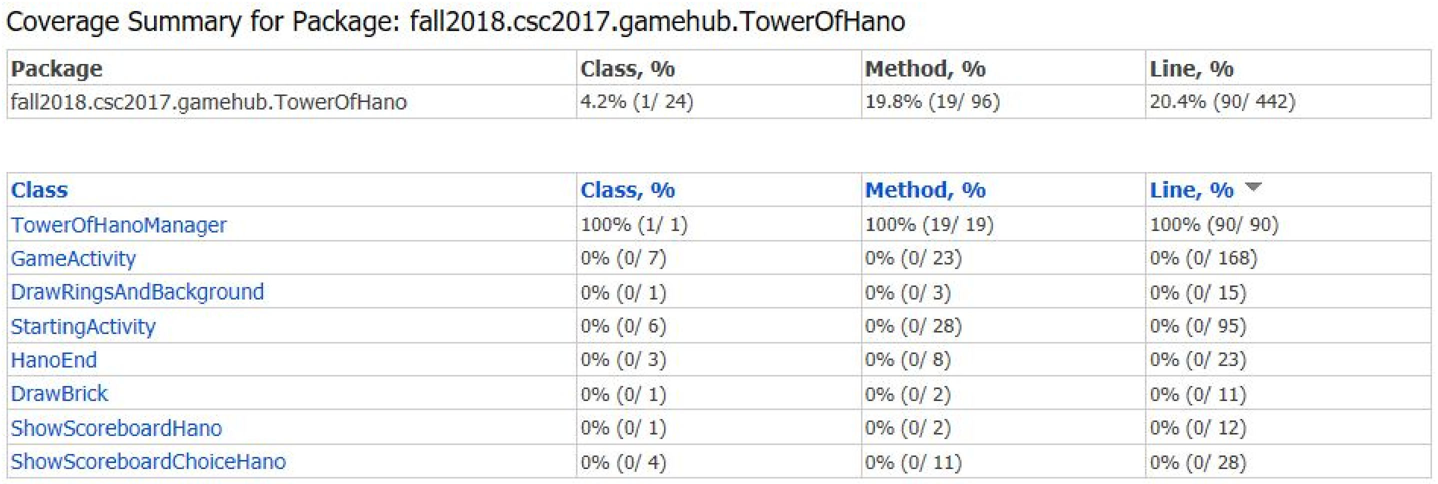
Coverage Summary for Package: fall2018.csc2017.gamehub.MatchingGame

Package	Class, %	Method, %	Line, %
fall2018.csc2017.gamehub.MatchingGame	18.2% (4/ 22)	28.9% (33/ 114)	38.9% (176/ 452)

Class ▲	Class, %	Method, %	Line, %
BoardManagerMatch	50% (1/ 2)	71.4% (10/ 14)	81.8% (45/ 55)
BoardMatch	100% (2/ 2)	100% (14/ 14)	100% (39/ 39)
Card	100% (1/ 1)	100% (9/ 9)	100% (92/ 92)
GameActivityMatch	0% (0/ 3)	0% (0/ 19)	0% (0/ 98)
GestureDetectGridViewMatch	0% (0/ 2)	0% (0/ 12)	0% (0/ 36)
MatchingEnd	0% (0/ 3)	0% (0/ 8)	0% (0/ 19)
MovementControllerMatch	0% (0/ 1)	0% (0/ 3)	0% (0/ 8)
StartingActivityMatch	0% (0/ 5)	0% (0/ 26)	0% (0/ 84)
scoreboardChoiceMatch	0% (0/ 3)	0% (0/ 9)	0% (0/ 21)

Note: BoardManagerMatch not fully covered due to usage of an android method to delay the flipping back of the card which requires a mock.

TowerHanoiTest

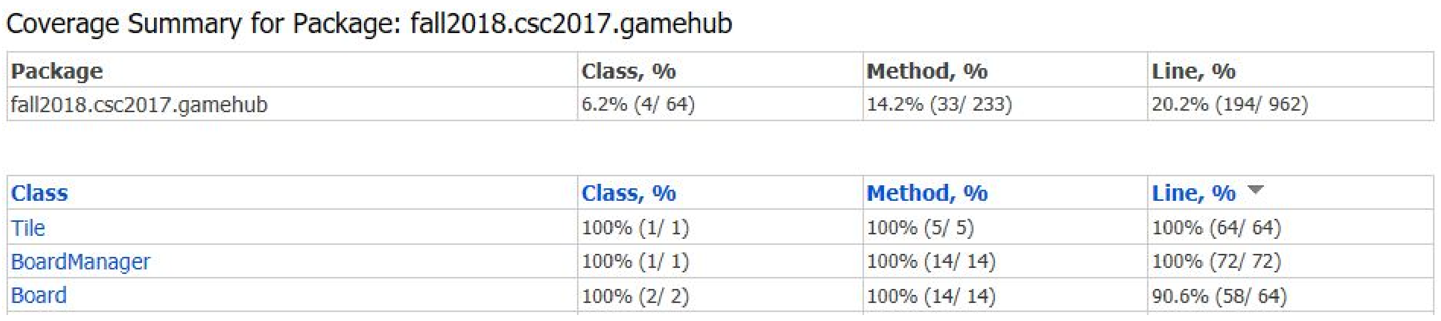


Coverage Summary for Package: fall2018.csc2017.gamehub.TowerOfHano

Package	Class, %	Method, %	Line, %
fall2018.csc2017.gamehub.TowerOfHano	4.2% (1/ 24)	19.8% (19/ 96)	20.4% (90/ 442)

Class	Class, %	Method, %	Line, % ▼
TowerOfHanoManager	100% (1/ 1)	100% (19/ 19)	100% (90/ 90)
GameActivity	0% (0/ 7)	0% (0/ 23)	0% (0/ 168)
DrawRingsAndBackground	0% (0/ 1)	0% (0/ 3)	0% (0/ 15)
StartingActivity	0% (0/ 6)	0% (0/ 28)	0% (0/ 95)
HanoEnd	0% (0/ 3)	0% (0/ 8)	0% (0/ 23)
DrawBrick	0% (0/ 1)	0% (0/ 2)	0% (0/ 11)
ShowScoreboardHano	0% (0/ 1)	0% (0/ 2)	0% (0/ 12)
ShowScoreboardChoiceHano	0% (0/ 4)	0% (0/ 11)	0% (0/ 28)

SlidingTilesGameTest



Coverage Summary for Package: fall2018.csc2017.gamehub

Package	Class, %	Method, %	Line, %
fall2018.csc2017.gamehub	6.2% (4/ 64)	14.2% (33/ 233)	20.2% (194/ 962)

Class	Class, %	Method, %	Line, % ▼
Tile	100% (1/ 1)	100% (5/ 5)	100% (64/ 64)
BoardManager	100% (1/ 1)	100% (14/ 14)	100% (72/ 72)
Board	100% (2/ 2)	100% (14/ 14)	90.6% (58/ 64)

Note: Board is not fully covered is because the implementation for the solvable sliding tile has lines

Scoreboard

How it's designed and where scores are stored

- Scoreboards are managed by ScoreboardManager
- All scoreboards are stored on one file in a HashMap.
- The key of the HashMap is the game and type (ex. "tile3x3", "match4x4") and the value is the actual scoreboard
- Scoreboards are stored in an ArrayList that stores a bunch of objects called a DataTuple
- Each DataTuple contains the username and the user's score

How they are displayed:

- Each game in the StartingActivity has a scoreboard button that goes to another page with options for them to view scoreboards of different type for the specific games
- User selects which type of the specific game they want scores viewed for and it will go to a custom listview of the scores.
- First column is the username and second column is the score/time in the listview.
- The top 10 scores of that specific game and type are displayed. This is a global high score (so includes all users who have played the game).

Design Patterns Used

Iterator Pattern

- Used in Board, BoardMatch, ScoreBoard
- Allows for traversal throughout the whole object (mainly arrays in this case) 1 by 1
- Faster and cleaner than just for loops

Observer Pattern

- Examples:
    - GameActivity observes Board in Sliding Tiles Game
    - GameActivityMatch observes BoardMatch in Matching Game
- The class being observed doesn't need to know anything about observers
- Reduce coupling with classes

Private Class Data - Structural Pattern

- Examples: User and DataTuple
- User and DataTuple are the data classes that store user information and cores. They are stored as private variables that can only be accessed by getters and setters.
- Protects attributes of the class from being directly manipulated reducing chance it is changed in a way that will break the code.

SOLID Principles Used:

Single responsibility principle

- Some examples: Tile, Card, Scoreboard, User
- Responsible for only one part of the app as data and functions related to responsibility is encapsulated in that one class. Makes code easier to understand and change as it is broken down.

Open/closed principle

- Example: CustomAdapter and ScoreListAdapter

- Allows for creation of a custom adapter that extends the classes that are built in without modifying the existing classes

Interface segregation principle

- Example: IStartingActivity, IEndingActivity & ISaveLoadFile interfaces and their associated classes
- Ensures that classes that have similar kinds responsibilities implement all required methods to perform responsibilities but with smaller broken down interfaces doesn’t force certain methods not required.
- In team programming it ensures that the individual who is programming the class implements all methods discussed/agreed upon by the team. Allows those writing classes that are going to interact with the class to know what methods will be provided by the class, what are the parameters required to use a method and the return value type.

Model-View-Controller Design

- Encapsulates layout and graphic elements of the code (things you see on the screen such as buttons) separate from the data and methods for processing the data.
- Breaks down code into smaller classes for better readability and also allows us to delegate task without having one class affect another.