WALKTHROUGH

Group 0610

WHAT EACH OF US COVERED OVER THE PROJECT

Tanvir Hyder: Designed 2048 game along with Kevin. Designed most of the UI. Score and Level. Designed score board and personal score board along with Jimbo. Implemented GameBoard generic class. Implemented feature level and xp.
Kevin Yang: Implemented the 2048 along with Tanvir, and have it be acceptable by interfaces and parent classes. Added ability to save/load for 2048. Implemented lots of game screens. Implemented hashmaps for account info.
Alex: Along with Jimbo, Implemented Checkers. Was responsible for lots of its logic and design, such as design in Checker's Manager, Gameactivity, Undo.
Jimbo: Along with Alex, implemented Checkers. Was responsible for CheckerBoard and filling in some methods, such as making of the board, autosave. Made Scoreboard, parent classes, and interfaces to be acceptable by all games.
Andries: Made sliding tiles always solvable. Implemented unittests for silding tiles. Made tile abstract.

IMPORTANT CLASSES:

1) GameBoard<T>
    - a generic class used as a parent class for all 3 game's boards and extensible for other board based game
    - The class uses generics of T to represent the different child Tile classes such as Tile2048, Tile (sliding tiles), CheckerTile
    - Uses a 2d array to hold the tile objects
    - For methods we have getTileAt which returns the tile object at specified rows and column, several getters which returns the row and column information and most importantly the iterator which saves us from code redundancy
    - Each of our game uses child Board classes that extends GameBoard with T being the corresponding child Tile class.

2) ParentTile
    - A parent class with properties id and background. The constructor takes in id as param which stands for the actual value of the tile. Ex: in 2048 a tile of value 4 has id 4.
    - The child tile classes reflect more on the functionalities of the game and sets the background image reference int using an array which stores R.drawable reference ints. Note that the child classes for different games uses different array for their R.drawable reference ints.

3) ManagerInterface
- An interface that all BoardManagers implement
- Key methods are gameFinished as well as isValidMove, they're implemented to make sure the game ending works correctly, as well as whether a move should be processed
- touchMove is also important, as it's the method that applies what is done in a move

4) GestureDetectGridView classes

- A key part to the detecting the correct movement for the program
- Allows sliding to be detected for 2048, and tapping for checkers and sliding tiles
- Works with the movement controller to send valid moves to boardManager to be made

UNIT TEST COVERAGE

Our unit tests cover the functionalities of all three games. Checkers functionalities and intricacies are thoroughly tested. For sliding tiles various features of the game are tested beyond the main tests (isValidMove, can you win). Undo is tested, as well as different complexities. For 2048, many main tests are included. This includes winning tests, losing tests, valid move tests, making sure you can make valid moves in every direction, making sure score updates accordingly, and combining works. Our tests also cover account to make sure that score integration works. Checkers is thoroughly tested, testing functionalities of isvalidmove, testing piece taking, testing forced capture, testing prime (multiple captures in a row) captures, testing kinging among other functionalities.

DESIGN PATTERNS

1) Observer pattern: This pattern was used to know when the view should update the board. The board itself with the tiles is observable, and the game activities were the observers. The view (GameActivity classes) would know when to update the tiles when the board sent them a notifyObservers() call. This method is invoked under board methods that are called when moves are made. For example, in sliding tiles it's called under swapTiles() as when a move is made from boardManager, that method from board is called.

2) MVC: This pattern was used in a couple of places, for example the implementation of 2048 (the other two games also make use of MVC). The model would be Board2048 and BoardManager2048, the view being GameActivity2048, and controller being GestureDetectGridView2048 and MovementController2048. By using MVC, it helped make testing easier and helped reduce coupling. It also made saving between different games easier, as you can save ManagerInterface which is implemented by all board managers.

LEADERBOARD MECHANISM

1) **Saving the score**: A score is initialized in boardManager and is changed through methods such as touchMove in the boardManager. Due to the broad definition of score, the score is also altered through gameActivity for that specific game. The score is saved to an account upon passing the win condition of the game Ex: gameOver or gameFinishes. In gameActivity we use the method getBoardScore to display and save the score. The user's account file is updated and the allAccounts file is overwritten with the updated account with the new score. This is done by serializing and deserializing the account (user) and allAccounts (user account array) files.

2) **DISPLAYING THE SCORE**: The class reads the allAccountsFile and collects the top score for the game achieved by a user and adds to an arraylist. The arrayList is then sorted by comparison of the accounts' scores. The scoreboardactivity uses passable variables (Intent.putExtra) to be used for different game scoreboards, which includes "currentGame" and "highToLow". currentGame is one of the three game constants, and is passed into Account.getHighScore(currentGame) to return the score of the corresponding game, and highToLow is determining how to compare scores (true = reverse comparison by times -1)
The method fillDisplayValues only reads the top 10 values of the array and displays them through the activity

GAME FUNCTIONALITY

1) **2048:** The game uses method touchMove() and isValidMove() to process a swipe and a valid move. There are 4 helper methods used by touchMove() : adjustTilesRow, adjustTilesCol, combineDoublesRow, combineDoublesCol. When a user swipes in a direction, the game first shifts the tiles by checking for empty tiles in the given direction. Then it iterates through all the row or column to check pairs of equal id tiles and merges them. It finally concludes by shifting tiles again to fill empty spaces in the given direction. IsValidMove() makes a clone of the current board state and applies the above adjustments to the clone. It then compares both boards and returns true if a change is found, false otherwise.
2) **SlidingTiles:** The game uses method touchMove and isValidMove() to process a tap. touchMove() given a position checks 4 surround tiles to see if any of them are blank. If a blank tile is found swapTiles() method is called to swap. IsValidMove() is similarly implemented but is used in gameActivity to make toast to user indicating invalid moves.
3) **Checkers:** The board is created with 3 rows of red checkers on top, 2 rows of blank, then 3 rows of black checkers. There are two stages to a move: A highlight tap, and a move tap, both processed by isValidTap and touchMove. Your first tap is on a checker, and it will highlight your available moves. isValidTap will detect if a boolean is true or not. This boolean determines whether you should be tapping on a checker or the spot you want to move to. A valid move is determined by whether there's a checker diagonally across. processMove is a helper method that is called to process every move. For example, eating checkers, kinging, checking for multiple moves in one turn and the like. In addition, you must capture if one of your checkers can. findCapture will return a list of checkers that can and must capture something during your turn. A normal black checker will find moves in addMovesForward, while a normal red checker will find moves in addMovesBackward (due to the orientation of the board. A kinged checker will check moves using both methods, regardless of colour.

LEVEL and XP feature:

- Level and xp reflects the time spent playing the different games on our app
- Every 100 xp, the user's account levels up.

- Both variables are stored in the user’s account
- For Sliding Tiles due to its difficulty, user gains 5 xp per valid move and 150 xp for solving the puzzle
- For 2048 game, user gains 125 xp per win (achieving 2048) and 50 xp when a game Is over
- For Checkers the signed in user gains 200 xp per win and none otherwise