1. Outline of Presentation
   - Self introduction of each group member (and responsible part for each one)
   - Brief walkthrough for the structure of our app ”Game Box!”(UML provided)
   - Specific demonstration when running for our app
   - Unittest coverage demonstration with 2 classes(MatchingCard,)
2. Design Pattern we use
   - MVC design pattern

     GameCenter:

     (M: Profile -- V: GameCentreActivity -- C: GameCentreController)

     UserHistory:

     (M: Account --V: UserHistoryActivity -- C: UserHistoryController),

     SavedGames:

     (M: GameManager --V: SavedGamesActivity -- C: SavedGamesController),

     GameStarting:

     (M: Constructor -- V: StartingActivity of 3 games -- C: GameStartController)

   Problem solving: each model classes know nothing about the view and controller classes. Activity classes only contain view setting methods and controller connects view and model together.

   - MVVM design pattern

     (M: Slidingtiles, MatchingCards, Game2048

     V: GestureDetectedGridView

     VeiwController: MovementController

     VM: SlidingtileGameActivity, MatchingCardsGameActivity, Game2048Activity)

   Problem solving: each backend model classes know nothing about the view, viewmodel and view's controller classes. Activity classes connects view and updated view according to user's behavior as a view model, and controller classes control the main view updates.

   - Observer pattern for 3 games:

   Ex:

   (Observer: MatchingCardsGameActivity -- Observable: MatchingBoard)

   (Observer: SlidingTileGameActivity – Observable: SlidingTileBoard)

   (Observer: Game2048Activity – Observable: Game2048Board)

   Problem solving: An object can notify other objects without making assumptions about what these objects are.

   - Iterator design pattern for 3 games:

   Ex:

   (Iterator: Board)

   Problem solving: a neat way to iterate over elements of the container. Besides, we can have multiple and independent iterators over the elements of the container. Also, it will not expose how the elements are stored in the iterator.

3. unit test coverage: ~ 20%

4. Scoreboard Implementation

   - We use a custom class called ScoreRecord which contains emails of users who belong to each record. And ScoreBoard can display the userNames with their records under the specific game. Then inside the class Global Scoreboard, we have created a private HashMap that has keys storing the gameId, and each value is a list of scoreRecords. The reason why we choose to save the gameID, is that we wish to implement a feature where other users wish to play the same game with same initial board, and see who can get the higher score(Slidingtiles for example does depending on a bit luck depending on how hard your initial board is). But now they can go into the game through UserHistory and click on restart for the same board as the first player started(which is another game feature).
   - We use the spinner to separate each game's score to different listview each spinner item corresponding to each gametype. When useres press the gametype they want, spinner will update the listview by getting data from the backend logic – it will retrieve the list of scoreRecords and extract the game then calculate the score out from it.

5. Important classes for current games AND new games:
   - Account, AccountManager, GameManager, CurrentAccountController
   - Board(depends on if the new game needs a board)
   - Game, GameFeature
   - Tile(depends on if the new game needs tiles)
   - GameScoreBoard, GlobalScoreBoard, UserHistory
   - CurrentAccountController
   - GameActivity, GameStartingActivity
   - SavedGamesController