Participants
==========
Nave Farjun 312494206 Navefarj@gmail.com
Dan Nechushtan 304990583 Dannech91@gmail.com

General
==========
Our design model gives the class Game the authority for most of the game main-events and logics - while other components - do 'minimal work' - such as rendering themselves according to the data - transmitted by the Game component .

This mechanism will allow us to create a multiplayer game, which will be managed by a single components (game). Without worrying about synchronizing between lower components - among different players.

The domino images are numbered according to a defined convention that allows the "binding" of a domino-tile image file to its model entity.

Classes
==========

1. Game - this is the main class, responsible for the data, logics, Assemble all the other components, and responsible for the flow of the game entities. Manage History, Manage the vector of Players.

2. Player - Mainly, holds the Player's Tiles. Named "myTiles". Acts as a stack. (Note: Game has the vector of Players. (and will synchronized between them in a multiplayer game - in the future).

3. Tile - represent a Domino Stone. Tile Identified by a unique key, and has a method that generates the corresponding image file name, to allow binding of a Tile class to its image file. Tile component contains function - which handle onClick event according to the Game behaviour .
Tile can be found in the Cash, Player's deck - (which are both - stack of Tiles) , or At the Board.

4.TilesCash - represent Cash itself. Acts as a stack. Main Responsible is giving a random card to a player, by using the function - 'bringTile'.

5. Statistics - responsible for the game Statistics. Differentiate between Global statistics parameters such as Timer (clock), and the player's specific statistics data for a single player. (Due to lack of time - In the future - every player will have its own statistics built-in in player component - while Global statistic will be displayed separately.)

6. Board - responsible for the Board. Gets its initial size (9x9) , and responsible for the resized according to game logics.

Notable Functions:
==========

1. bringTileFromCash() - Considered as a single move of player in the game. invoked by UI onClick. button - “Give me Tile!”. Responsible updating the History, choosing free random tile from cash, and updating the current player deck of tiles + score.

2. handleOnClickTable() - Considered as a single move of player in the game. invoked when a player decide to put specific tile on the board. Gives the Tile the right position, check if Game overed, update history,deck,statistic, and other relevant components.
3. prev() & next() - Display only at the end of Game. Enable to restore the game through all its positions. The function are accessing the history back and forth.
4. startNewGame() - Correspond to UI button ‘restart’. Invoked on click event. Touches almost all the components. Initial Board, Cash, player’s Deck, Delete History, reset clock, player’s statistics and other relevant flags.

Assumptions
==========
1. The game is for 1 computer player, and 1 human player.

General Remarks
==========
1. We chose to help the user understand what are the legal squares to put the tile in - by painting -only the valid cells - background in blue.
2. We store relevant data (in History vector) as a non-reference data. We called it ‘deepCopy’ - In order to avoid changes in data which already stored in the history. DeepCopy is done by JASON parse and stringify.
3. Each Tiles in the board gets attribute named ‘position’ which decide whether a tile will be displayed Horizontal or vertical. Position calculated by Game - when tile placed in Board.

Bonus:
==========
1) Undo - Enable the player to ‘undo’ - the last move.