## Due:

April 8, 2019, **before 11:59 pm.**

Updated: Mar 25. Fixed description of Model methods.

## Notes

- Please follow both the "Programming Standards" and "Assignment Guidelines" for all work you submit. Programming standards 1-25 are in effect.
- Hand-in will be via the D2L Dropbox facility. Make sure you leave enough time before the deadline to ensure your hand-in works properly.
- All assignments in this course carry an equal weight.

# Assignment 4: whodunit?

## Description

In this assignment you will be implementing *"whodunit?"*, a game that involves deduction to determine the "who, where and how" of a crime. Each player makes guesses about the crime, and then other players refute that guess, if possible, based on information they have on cards that have been dealt to them. The game "*whodunit*?" is mildly similar to a well-known boardgame called "Clue". For a three minute introduction to the board game Clue, please see this video: https://www.youtube.com/watch?v=sg_57S4l5Ng

You will implement the game in ruby. It will be a text based game that allows one human player to play against a group of computer players.

## Game Simplifications

"*whodunit*?" is simpler than Clue. Most notably, there will not be a game board or dice in "*whodunit*?", so there is no movement in the game. Additionally, the players do not take on character identities in the game – players and suspects are separate in "*whodunit*?". The game will thus consist of players taking turns making guesses about the identity of the murderer, the weapon and the location of the murder. The first person to guess correctly wins the game. Also in "*whodunit*?", a turn consists of either a suggestion and an accusation (not both).

## Major Assignment Parts

There are two major parts to the game.

1. **Players**: you must create two classes for game players. The classes should be part of a hierarchy with an abstract class at the top of the hierarchy. The concrete subclasses are the Players that plays the game. You are responsible for implementing one subclass for human players and one subclass for a computer player.

2. **The Model**: this class manages all of the players of the game, all the cards in the game and the turns of the board. You will write this class.

The main method (provided to you) will create players, and a model, and then call the model to start a game. It will allow one human player to play against a group of computer players.

## Basic Classes

There are two helper classes that help the game:

1. **Card**: a card is one of the cards of the game. It has two fields: a type and a value.
2. **Guess**: a guess is made by a player when they want to learn more information about the game. A guess consists of three cards, and a boolean variable for whether the guess is a suggestion or an accusation[1]. The names of the fields should be "person", "place" and "weapon".

These classes should have initializers that accept the required information. They should also have getters (attr_reader) for the information, but no setters. Use ruby symbols (`:person`, `:place`, and `:weapon`) to define the type of a card.

## Model

The model is the portion of the program that stores all of the players, all of the cards, and runs the game. The model will give all players a number between 0 and n-1 where n is the total number of players. Players are arranged in order 0 through n-1 around the table. That is, after player i's turn, it will be player (i+1)%n's turn.

The model will create, shuffle and distribute all of the cards to the players, as well as holding the three "answer" cards (the who-where-how cards).

The model is required to have the following methods:

~~1. A method **setPeople** that takes one parameter: an array of individuals that are suspects. These should be objects of type Card.~~
~~2. Two methods **setPlaces** and **setWeapons** that work the same as setPeople, but for places and weapons.~~

1. The Model class should have an **initializer** that accepts three arrays as parameters: the array of all suspects, the array of all locations and the array of all weapons, in this order.
2. A method **setPlayers** that takes one parameter: an array of players in the game. The players should be initialized by this method, including calling the setup method (see #1 below in the Players description) for each player. The method also assigns indices (see the setup method in #1 below) to each player. Indices can be assigned to players arbitrarily.
3. A method **setupCards** that does all of the setup for cards before a game starts. This method takes no parameters. This method should choose the answer for the game (the criminal, the location and the weapon) and then distribute all remaining cards to the players. Similar to the original board game, cards are distributed to players based on their index (i.e., player 0, then player 1, then player 2, … ) and it is ok if some players receive one more card than other players.

[1] In the game, an accusation is the final guess of the game for a player – if they are correct, they win the game. If they are incorrect, they are eliminated from the game.

4. A method **play** that runs the game. This method takes no parameters. See the pseudocode in the **Game Pseudocode** section below to describe how the game is run.

## Players

Each player will have the following methods that allow the player to make a turn and receive information about the game:

1. A method **setup** that receives five parameters: the number of players in the game, the index of the current player (what player number am I?), a list of all the suspects, a list of all the locations and a list of all the weapons.
2. A method **setCard** which indicates that the player has been dealt a particular card.
3. A method **canAnswer** which has two parameters: a player index `i` and a guess `g`. This method represents that player `i` (which is different from the current player) has made guess `g`. The current player is responsible for "answering" that guess, if possible. The method should either return a card (which the current player has it their hand) or nil, to represent that the current player cannot answer that guess.
4. A method **getGuess** (no parameters). If this method is called, it indicates that it is the current player's turn. The method should return the current player's guess for that turn. The guess may be a suggestion or an accusation.
5. A method **receiveInfo** which has two parameters: a player index `i` and a card `c`. This represents that the current player has made a guess (previously) and the player at index `i` has one of the cards of the guess, `c`. If no player can answer any information about the current player's previous guess, then `c` is nil and `i=-1`. Otherwise, `c` is guaranteed to be a card from the current player's previous guess and `i` is a valid index. This method is guaranteed to only be called after a call to getGuess by the current player.

The two classes should have specific names:

1. The human player class should be called InteractivePlayer.
2. The computer player class should simply be called Player.

The human player class should prompt the user for information. For instance, the method canAnswer should show the human player the current guess and the current index i, show the player the cards they have that they can show player i, and then ask which card they wish to show. The program is not responsible for managing the human player's deduction (i.e., it does not need to track which cards they have been show by other players, etc.).

You must also implement a computer player. The players should both obey the requirements that it makes reasonable guesses (i.e., it never makes guesses about cards it is already aware of, if possible). To ensure that the computer player is making reasonable guesses, it must pass the ruby unit test file that is provided to you.

## Game Pseudocode

Here is the pseudocode for how the game progresses. Note that this pseudocode does not necessarily include details on every call to the players' methods.

```
while game is not over:
  ask active player for their guess
  if guess is an accusation:
    test if accusation is correct:
      if correct, game is over, active player is the winner.
      if not correct, active player is removed from the game.
      (game is over if there is only one player remaining.)
  else:
    ask players if they can respond to the guess.
      (starting with next player after active player)
    if another player can answer the guess:
       provide answer to active player
    else:
       report that to the active player
  if game has not ended:
    move to next active player.
```

## Data Structures

In this assignment, you are not required to build you own data structures. You can use ruby arrays in this assignment.

## Small Hints

1. You may find the method .shuffle for arrays helpful for shuffling cards before dealing. The method returns a shuffled version of the array.
2. The method .upcase converts a string to uppercase
3. You may want to read about Hash, an associative array in ruby that lets you index by objects other than integers.
4. A file of text prompts is given to you to help you write your messages to the user.

## Submission

Submit all of your code through D2L. Submit all of your ruby files as one zip file, including a README file describing how to run your program.