# CS 371 Laboratory:

# Game Framework Tutorial

This laboratory will give you practice using Prof Vegdahl's game framework by implementing a simple game called Pig.

As you work through the exercises in this laboratory, you should only modify code in the pig package. Do not change the Game Framework code, which is in the **game** package.

In this laboratory, some instructions have been left deliberately high level so that you will discover the details yourself. In particular, pay attention to methods and variables that are provided by parent classes in the game framework.

## How to Play Pig

Pig is a game for two players who use one or more dice. For this lab, we will use exactly one die. Each player has a score that starts at zero. The players take turns trying to increase their score. The first person to reach a score of 50 is the winner.

A player begins each turn by rolling the die. If the player rolls a 1, then his turn ends immediately, and the score remains unchanged. If the player rolls another a number, he can choose to HOLD or ROLL.

If he chooses to HOLD, the die will be added to his score and his turn ends.

If he chooses to ROLL, then the die is rolled again. If he rolls a number other than a 1, then the number rolled is added to the previous number(s) rolled. This total of all successful rolls will be added to his score when he chooses to HOLD.

A player can ROLL has many times as he likes, trying to acquire a greater and greater increase to his score. However, the instant he rolls a 1, all is lost.

## Preliminaries

### Retrieve the Starter Code and create a new Git Repository

Create a new project by cloning the following git repository:

### *https://github.com/cs301up/PigGameStarter*

~~Then upload your starter code into a new GitHub project. Add your lab partner as a contributor to the project and verify your partner has access to upload changes to this project as a result.~~ (This is not necessary this term since both of you are working on separate copies instead of one code base.)

When you run the project, if it gives you a “Sync Android SDKs” pop-up, press “OK”.

### Familiarize Yourself with the Project

View the layout file for this game: app/res/layout/pig\_layout.xml. This is what the game will look like during play. The user will see her current score, her opponent's score, and the current total of successful dice rolls that will be added to her score if she chooses to HOLD.

The layout also has a HOLD button and an image button that shows the current die roll and can be pressed to ROLL again.

The project already contains the game framework and a Java package named **pig** which contains some preliminary files. Most of these files are just stubs waiting for you to complete them. PigMainActivity already contains code to setup a default game configuration. PigHumanPlayer has some code to initialize the GUI.

As a result, when you run the starter code you can configure and start a game of Pig and the GUI for the human player will be displayed. However, pressing the buttons has no effect. It's time to get to work!

## Laboratory

### Part 1: Create PigGameState and action classes

Create a new class in the pig package named PigGameState. This class should extend the GameState class in the edu.up.cs301.game.infoMsg package.

PigGameState will contain all the information about the current state of the game. In particular, this state must have instance variables that store the following information:

* the id of the player whose turn it is (0 or 1)
* player 0's score
* player 1's score
* the current *running total* that would be added to the current player's turn if the HOLD action is taken
* the current value on the die (1 through 6)

The game framework expects player IDs to start at 0, so use 0 and 1 (not 1 and 2). Similarly, name your score variables something like player0Score and player1Score.

Your class will need the following methods:

* A no-argument constructor that initializes the state appropriately for a new game
* A copy constructor that creates a new instance that has the same values for its variables as a given instance
* A getter (accessor) method for each instance variable
* A setter method for each instance variable

(Note: the official game state is kept as a private instance variable in the PigLocalGame object. It cannot be corrupted because the PigLocalGame always creates a copy of its state whenever it shared it with another object.)

Nominally, a player will have two actions while playing Pig: HOLD and ROLL. Create a new class in the pig package to represent each of these actions: PigHoldAction and PigRollAction. Each of these classes will inherit from the GameAction in the edu.up.cs301.game.actionMsg package. Note that an instance of one of these classes merely represents an action. It does not actually take that action. The PigLocalGame class will be responsible for this (see below).

This parent class constructor requires that a reference to a GamePlayer object representing the player who is taking the action be specified. Therefore, each of your classes must require this information as well to pass on to the parent.

**checkpoint 1 (20 points): Have your instructor or lab assistant verify that your PigGameState and action classes are properly implemented.**

### Part 2: Complete the PigLocalGame Class

As configured, Pig can be played with either one or two players. The PigLocalGame can know the number of players in the game by looking at the length of the inherited array that contains the names of all the players.

The local game class is the central hub for the game. It sends game states to players and receives actions from them. Complete the stubbed-out methods in the PigLocalGame class as follows:

* Create an instance variable that contains a reference to the official state of the game (a PigGameState object)
* The constructor should initialize the game state instance variable by instantiating a new PigGameState.
* The canMove method should determine if the player id given to it matches the id of the player whose turn it is according to the game state
* The makeMove method must perform several tasks:
  + Use the instanceof operator to determine whether the given action is a HOLD or ROLL action. (If it is neither, do nothing and return *false*, indicating that the move was illegal.)
  + If it is a HOLD action:
    - Add the *current running total* to the score of the player whose turn it is.
    - Set the *current running total* to zero
    - If there is more than one player in the game, make it the other player’s turn.
    - Return *true*, indicating that the move was legal.
  + If it is ROLL action:
    - Roll the dice. (That is, set the die value to be a random integer in the range 1..6.)
    - If the die value is anything except 1, add the die value to the *current running total*.
    - If the die value is 1, set the *current running total* to zero, and (if there is more than one player) make it the other player’s turn.
    - Return *true*, indicating that the move was legal.
* The sendUpdatedStateTo method should make a copy of the current game state and use the GamePlayer sendInfo method to send it to the given player object.
* The checkIfGameOver method should check to see if either player's score meets or exceeds 50 (the winning threshold). If the game is over, the message returned should include the name of the player who won, and the score that they achieved.

Run the debugger and verify that the receiveInfo method in PigHumanPlayer gets called when the **game** starts (not the setup screen). Since receiveInfo is currently empty, you will need to put a line of executable code in the method in order for the debugger to stop at a breakpoint within the method.

**checkpoint 2 (20 points): Have your instructor or lab assistant verify that your code is properly implemented and that the receiveInfo method in PigHumanPlayer is getting called.**

### Part 3: Complete the Computer Player

(Note: both the computer player and the human player can know their own player-number in the game by examining an inherited instance variable. They can also know the number of players and the names of the players by examining a different inherited instance variable.)

In the receiveInfo method in PigComputerPlayer, you will need access to the game state. Within the method, instantiate a new PigGameState with the given GameInfo object as a parameter that is typecast to a PigGameState.

The AI for this game need not be very smart. Implement the receiveInfo method in PigComputerPlayer so that:

* If it is not that player’s turn, it simply returns.
* If it is that player’s turn, it selects randomly (50-50 chance) between the HOLD and ROLL actions. To take an action, create an instance of the associated action class (PigHoldAction or PigRollAction) and send it to the game via the sendAction method. The computer player has access to the game via an instance variable declared in its parent class.

**checkpoint 3 (20 points): Have your instructor or lab assistant verify your implementation.**

**Part 4: Complete the Human Player and Debug Your Game**

The PigHumanPlayer class is already partially implemented. Complete the class by implementing these methods:

* The receiveInfo method should:
* Check to ensure that the *info* object is a *PigGameState* object. (It might not be if the player played out of turn, so that the game sent the player information about the error.) If it is not a *PigGameState* object, it should flash the screen and return. (Hint: there is an inherited method that flashes the screen.)
* Update the TextView objects to display your score, opponent score and *current running total* respectively. You can ignore the message text view for now. Note, Your Score should always reflect the human player’s score, regardless of whether the human is player 0 or player 1.
  + Hint: The compiler will accept a number passed to the setText method for a TextView object, but it will fail at run time.
* In addition, this method should update the image on dieImageButton to reflect the value of the die. These images have already been provided for you in the res/drawable folder as face1, face2, face3, face4, face5 and face6. To set the currently displayed image use the setImageResource method. For example, to display the face3 image use this command:

dieImageButton.setImageResource(R.drawable.face3);

* The onClick method is called whenever the user taps the HOLD button or the die button (for a ROLL). Your method must detect which button was pressed and send an appropriate action to the game just as you did for the computer player. Since the two buttons are different types, one way you can determine which button was pressed is by using the instanceof operator.
* In order to make it clearer who just played and what happened, incorporate a delay (e.g., two seconds) after the computer makes a move. That way, the human user can see each die roll as it occurs, rather than only seeing the cumulative effect.

At this point, your basic game is code complete, and you should actually be able to play Pig. Test and debug your code until you can play the game. Make sure it works in 1-player mode, where you are the only die roller; there should be no switching of turns in 1-player mode and the die roll value should always be added to your score. The game framework does not support 1-player mode for the computer, so you do not have to test that scenario. Also make sure the game works as expected in 2-player mode where the human is player 1 and the computer is player 2 and vice versa.

**checkpoint 4 (20 points): Have your instructor or lab assistant verify that your application runs properly. It should work in 1-player mode (human) and 2-player mode (human vs computer AND computer vs human).**

*The remaining checkpoints in this lab can be done in any order. You must complete at least one of them to receive full credit for the lab. Complete more than 1 to receive 10 points extra credit.*

**Part 5: Add a “Smart” Computer Player**

Create a new computer player that it is significantly better than random. When starting the game, it should be possible for a player to choose to play against either your new smart player or the original computer player you've already implemented. In addition, the **smart computer player should be the default choice**. To add the new computer player, you will have to modify the game configuration code in PigMainActivity.

Good decisions in Pig rely not only upon the number of points you would receive if you selected the HOLD action, but also on the difference between your score and your opponent's, and on the remaining number of points needed to win. **Your smarter computer player must be informed by these data as well**.

**checkpoint 5: You or your instructor or lab assistant should lose at least 1 game out of 3 to the computer, even when they are first to start. It should work in human vs computer AND computer vs human.**

**Part 6: Make the Game More User Friendly**

Modify the game so that it's clearer to the user what is going on. Specifically:

* There should be an indication on the GUI whose turn it is. For example, the background color of the die could be red if it’s the human player’s turn, and gray if it’s the opponent’s turn.
* At the end of each player's turn, the game should display a useful message to the human user about what happened (e.g., “Oh no! Alexa rolled a 1 and lost everything!” or “John added 11 points to the final score.”).
* Rather than labeling the scores as “Your Score” and “Opponent’s Score”, use the players’ actual names. Take care, as you will likely not know the names of the players when the PigHumanPlayer is created. You will need to set the label after the game and players have all been connected. (There is a method that you can override whose purpose is to do exactly that kind of initialization. Check the GameHumanPlayer class.)
* Update the winning message to include the players’ actual names instead of just Player 1 or Player 2.

To accomplish all of this, you may need to modify the game state to include the message that should be displayed to a human user. (*Hint:* Do not forget to update the copy constructor in PigGameState if you add any new instance variables to that class.)

**checkpoint 6: Have your instructor or lab assistant verify that your application runs properly.**

**Part 7: Finish up**

We can’t do the usual next part remote distanced, so we will end here.

**~~Make sure~~** ~~both partners have a copy of the completed project~~ **~~before~~** ~~you log off. Enjoy the rest of your day!~~

**~~Part 8: Ensure that the game can be played over the network~~**

~~Try playing the game over the network, using two human players, each with their own tablet:~~

* ~~One tablet runs the game as the server (“LOCAL GAME”), but selects “Network Player” as one of the players. (Make note of the tablet’s~~ *~~IP Code~~* ~~near the top-right corner. If it shows as “Unable to Determine”, it probably means that your tablet is not logged into the network.)~~
* ~~The other tablet (the client) runs as “REMOTE GAME”. Select “Local Human Player” as the player type, and enter the server’s~~ *~~IP Code~~*~~.~~
  + ~~Note: so that you don’t have to keep retyping the server’s IP Code each time, consider pressing “Save this Configuration as the Default” button after the IP Code is entered.~~
* ~~The server should press “Start Game!” first; then the client should press “Start Game!”.~~

~~If the game plays properly over the network, you get the points for very little work. However, it’s possible that things will not work right:~~

* ~~The human player over the network might not display things from her point of view. (E.g., it may list her score~~ *~~below~~* ~~her opponent’s.)~~
* ~~The game might crash if a player attempts to press a button before the network-based player has connected to the game. (Note: the game might be null before all the players connect.)~~
* ~~It’s possible that the~~ *~~state~~* ~~or~~ *~~move~~* ~~objects contain references to large objects (such as the entire game); this will likely cause the game to hang or crash.~~

**~~checkpoint 8 (15 points): Have your instructor or lab assistant verify that your application runs over the network, using two tablets.~~**