Information Expert
Yes. We have used this pattern. We tried to assign specific responsibilities to specific classes. For example, we tried to keep functionality like pawns in the posButton class, and background in the mainWindow class. The posButton class would know when the user moves and builds walls.

Creator
Yes. We have used that.The Menu instance was created by the MainWindow class. Because the MainWindow has the responsibility to create menu.

Low Coupling
Yes. Only MainWindow creates the girds, has nothing to do with other classes. We tried to separate functionalities and not cooperate with different classes.

High Cohesion
Yes. For example, we tried to keep all the grid's operation methods like changing colours (represent movement) inside the PosButton class.

Controller. We create several session controllers. The controller handles what operation to perform (to build a wall or move the pawn) when the user clicked the mouse on specific places.

Polymorphism. Grids and spaces between grids are all buttons, but we wrap those as different objects but implement JButton.

Pure Fabrication. We are going to use this design pattern if we create the storage database for saving games.

Indirection. If we create saving functionality, then the functionality would go through an interface to save it in the database. And if we are trying to connect the network, the Internet interface should be provided.

Protected Variations. The methods inside our classes do not call the method from other classes. We also tried to keep the variables inside each class "private" to use this pattern. The user would not know what is happening when creating classes, so kind of like "abstraction."

Singleton. There is only one board and one window.

Adapter. If the user choose more players, we may use it.