

## PoolGame v3

### enter Konami code to activate extension features

The received code was not designed for extensibility. It did not apply interface segregation or dependency inversion, leading to an overpopulated interface often with non-virtual methods that do nothing. This is caused in particular by the Game class being shared between state1 and 2 implementation, even though it was clearly not written for that purpose. Another point of contention is the 'magic' MouseEventAble class, which couples its subclass with itself and dialog. It also makes it awkward to try and handle key events through a different interface, or to expand the interface to include key events. Implementation wise, there were a few bugs and some sections where drawing and animation is coupled when it appears explicitly separate to the client class. There were missing documentation on several classes, but the most severe offender is the MouseEventAble class which tells the reader to 'go away'. The code style was inconsistent - some constants were capitalised, some member variables did not follow the 'm\_name' convention.

To implement the undo functionality, the memento and prototype pattern was used. The memento allows the state of the game to be stored and restored, while the prototype allows objects to be created in the same state as the prototype. The memento pattern can lead to high memory usage and leaks if not implemented carefully. With respect to the code base, where states are often stored in a pointer, copying and moving data can cause memory fragmentation, slowing down the program. A drawback of the prototype pattern is that it pollutes the class hierarchy with the clone method, making it cluttered and obscuring other important functionality.

The Observer pattern was used for the undo function and the extension. By having the CueBall class notify its observers of successful mouse fire events, we were able to find out when something interesting happens on the CueBall (e.g., a move that should be stored on the caretaker of the memento) without maintaining an explicit reference to the object or inheriting the MouseEventAble class. The current implementation has the limitation that the observer cannot select which event to observe and information about the event can only be found from a reference to the subject passed in the update method. A common concern of this pattern is that the update method can be expensive and cause more updates. Given that the project is in its final stage and there's only one subject and two observers, this is not a major concern in the code base.

