Engineering of Software Subsystems SWEN-262

Observer Pattern "Mini Design"

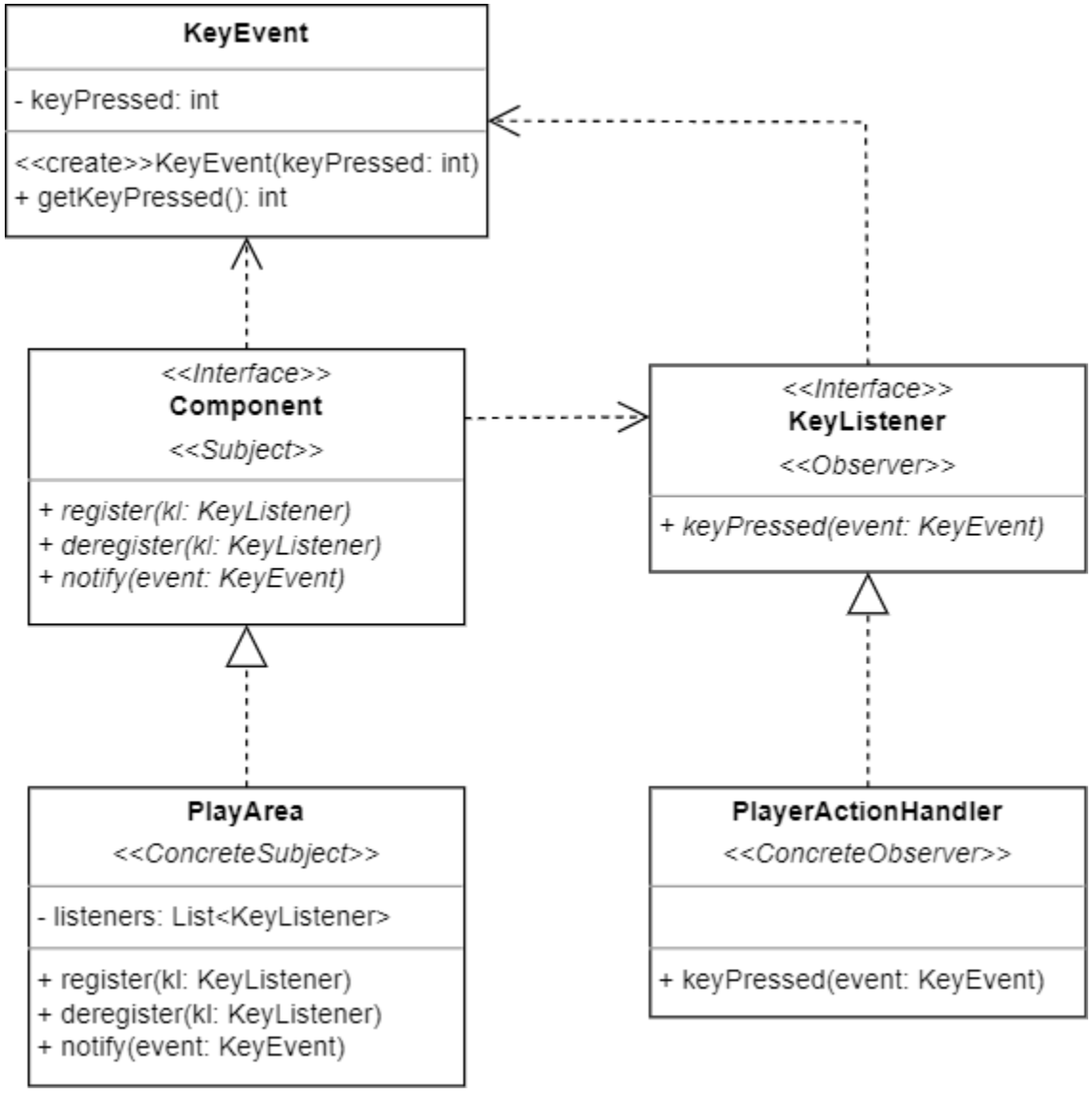
Rationale

The Invaders From Space™ requirements describe several features that require the application to respond to user input, including that "[t]he player uses left and right arrow keys to move their spaceship in the corresponding direction," and "the spacebar [is used] to fire a single shot." The requirements also state that "if at any time the player presses the 'ESC' key, the game pauses and they are prompted to quit. Pressing the 'y' key will quit the game. Pressing the 'ESC' key again will unpause" and that "the game should be as responsive as possible." The intent of the Observer pattern is to establish a dependency between a subject and its observer such that, when the state of the subject changes, the observer is updated automatically. The aforementioned features describe changes in state, i.e. keys being pressed by the user, and the actions that need to be taken in response, i.e. moving the ship, firing shots, or pausing/quitting the game. The observer pattern is an ideal match for this requirement.

The observer pattern includes several positive impacts on the overall design of the system. By defining interfaces for the Subject (Component) and Observer (KeyListener), the pattern inserts a layer of abstraction between the Real Subject (PlayArea) and its Concrete Observer(s) (PlayerActionHandler), thus exemplifying the dependency inversion principle. This separation is enabled by the dependency injection principle; the concrete observers are registered with the concrete subject by "injecting" them using the register method. The observer pattern also maintains high cohesion - each participant in the pattern has a specific, non-overlapping responsibility. If any additional key events need to be handled in future versions of the game, additional concrete observers may be added to the subsystem by implementing the KeyListener interface without altering the existing classes or interfaces, thus adhering to the open/closed principle.

One trade-off is that this implementation of the pattern comprises 4 small classes and interfaces, thus increasing the overall coupling in the system. However, this tradeoff is acceptable given the many strengths of the design. Separation of concerns could be improved in the current design; a single class handles the 4 different possible key events (move, shoot, quit), and it may be better to separate these into different classes. The team decided to keep the functionality together because the code to handle each separate event is very small, but this decision may be revisited in the future.

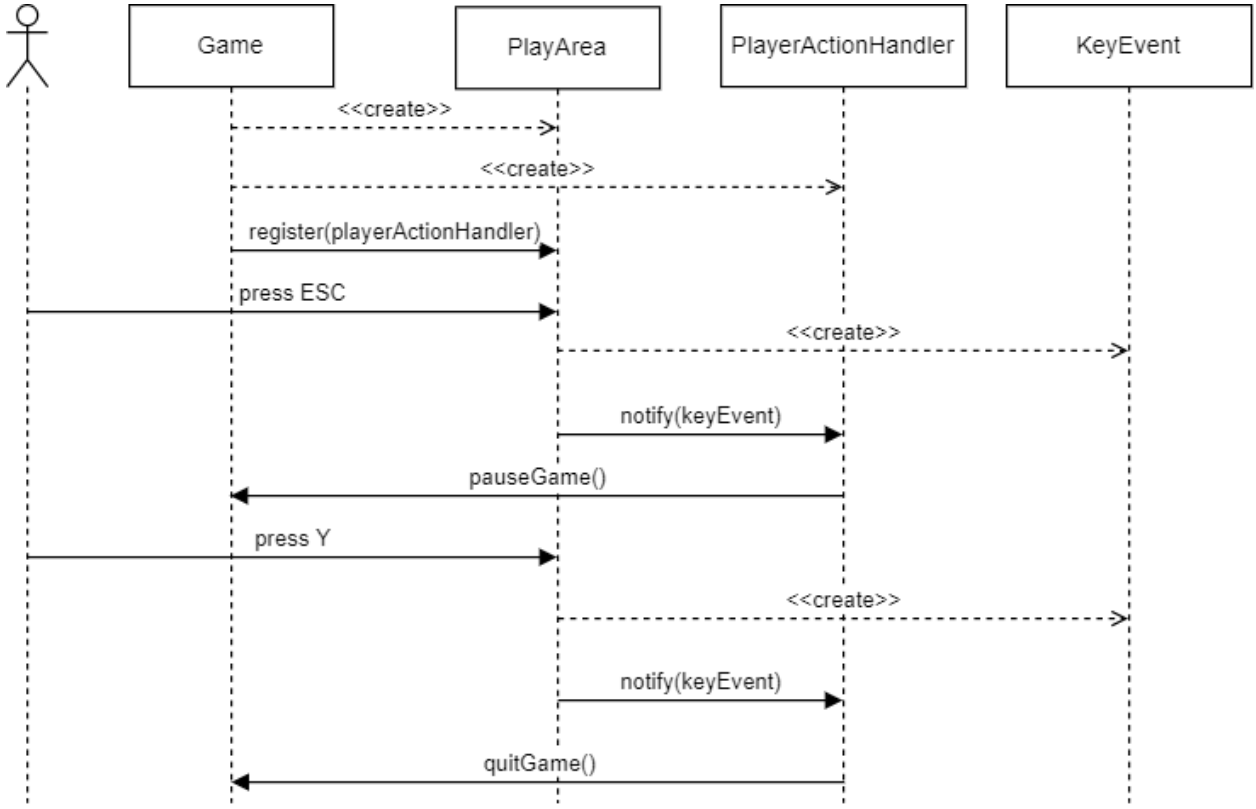
UML Class Diagram



GoF Pattern Card

Subsystem Name: Player Action Subsystem		GoF Pattern: Observer
Participants		
Class	Pattern Stereotype	Participant's contribution in the context of the application
Component	Subject	Defines the interface for any class that can be observed for key presses. This is most likely to be a GUI component of some kind, like a panel.
PlayArea	ConcreteSubject	The GUI component that displays the play area including space ship, aliens, etc. This component will have focus during play, and so will generate an event at any time that the user presses a key while the game is running.
KeyListener	Observer	The interface for any class that should be notified when the user presses a key on an observed subject. There may be several such listeners in the game.
PlayerActionHandler	ConcreteObserver	Interprets key presses from the player into actions in the game, i.e. left arrow moves the space ship left, right arrow moves right, and so on.
Deviations from the standard pattern: None		
Requirements being covered: 1a. Ship movement, 1b. Firing weapons, 1c. Pause/quit, 2. Responsive to player input.		

Sequence Diagram



UML Sequence Diagram showing the "Quit Game" feature described in the requirements.