№ 3 Применение паттернов поведения

Задание

1) Внутри каждой игры есть код, отвечающий за считывание пользовательского ввода - нажатия на кнопки, клавиатурные события, нажатия мыши и т.д. Этот код обрабатывает ввод и преобразует его в соответствующие действия: jump, go, down и т.д.

   Используя паттерн **Command** добавьте две-три команды для управления объектами.

   Для этого создайте интерфейс, представляющий запускаемую игровую команду, например:

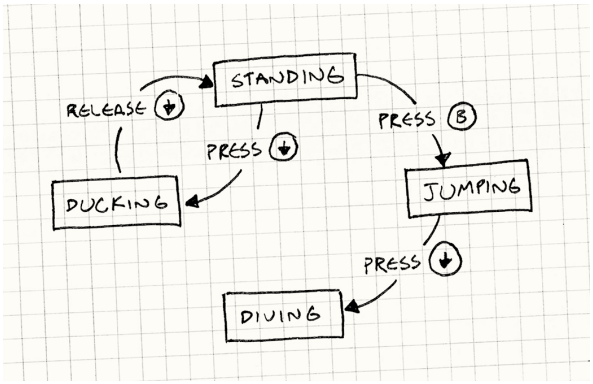
```
interface ICommand
{
    void Execute();
};
```

   Создайте классы для каждой из различных игровых команд:

```
class JumpCommand : ICommand
{
    public void Execute() { Jump(); }
};
class FireCommand : ICommand
{
    public void Execute() { FireGun(); }
};
// и т.д.
```

   В обработчике ввода храните ссылку на команду для каждой кнопки. Там, где раньше пользовательский ввод напрямую вызывал функции, теперь появился промежуточный слой косвенности:

2) Определите для вашего объекта несколько состояний. Нарисуйте на листе состояния в виде прямоугольников, а переходы между ними - линии со стрелками - управляющие команды. Что-то похожее на конечный автомат. Например, так (их может быть меньше, скажем два состояния):



Объект находится только в одном состоянии в каждый момент времени. Он не может стоят и двигаться одновременно. Нажатие кнопки приводит объект в другое состояние. Каждое состояние имеет набор переходов. Когда происходит пользовательский ввод, если он соответствует текущему состоянию, объект меняет свое состояние туда куда указывает стрелка.

1 вариант) Реализацию паттерна **State** начните с перечисления. Например:

```
enum State
    {
        STATE_STANDING,
        STATE_JUMPING,
        STATE_DUCKING,
        STATE_DIVING

              };
```

У объекта будет только одно состояние `_state`. В зависимости от ввода и текущего состояния происходит переход.

2 вариант) Это вариант лучше. Определите интерфейс **State** для состояния. Например,

```
interface ObjectState
    {
        void HandleInput(MyObject hero, Input input);
        void Update(MyObject  hero);

             };
```

Затем классы для каждого из состояний реализующий интерфейс. Методы определяют поведение в данном состоянии. Например,

```
  class DuckingState :  ObjectState
{
public DuckingState()    { }

   public  void HandleInput(MyObject hero, Input input)
    {
        if (input == _DOWN)
        {
            // Переход в состояние стояния...
            hero.setGraphics(IMAGE_STAND);
                hero.ChangeState(new JumpingState());
        }
        //...
    }

     public void Update(MyObject hero)
    {
        //...
    }


           };
```

Затем игровому объекту MyObject даем ссылку не текущее состояние и делегируем его работу состоянию:

```
class MyObject
  {
      public void HandleInput(Input input)
      {
         _state.HandleInput(this, input);
      }

      void Update()
      {
          _state.Update(this);
      }
      public void ChangeState(ObjectState state) { }
      // Другие методы...
      private ObjectState _state;
  };
```

Чтобы изменить состояние нужно чтобы `_state` ссылался на другой объект `ObjectState.` У вас получится паттерн **State**.

3) Добавьте команду сохранения состояния объекта. Для этого создайте класс Memento (реализует шаблон поведения **Memento**), который хранит информацию о состоянии объекта выбранного вами класса: состояние и два метода set и get. Добавьте в сохраняемый класс методы (создает Memento и сохраняет состояние объекта) и новый класс Restorer (восстанавливает сохраненное состояние).
4) Опционально добавьте паттерн: **Observer** для контроля достижений в игре, **Strategy** и т.д.

Вопросы

1. Назначение паттернов поведения?
2. Нарисуете диаграмму классов и поясните принцип работы паттерна Chain of responsibility. В каких случаях надо его применять?
3. Назначение и принцип организации паттерна Command. Поясните как он связан с конечными автоматами
4. Как реализовать паттерн Observer?
5. Нарисуйте диаграмму классов для паттерна Mediator. Поясните его назначение.
6. В чем разница между паттернами Mediator и Facade?
7. В чем суть паттерна Memento? Поясните на примере.
8. Расскажите о паттерне Visitor?
9. В каких случаях надо применять Null object?
10. Поясните на диаграмме классов как реализовать Strategy.
11. Перечислите и поясните принципы проектирования SOLID.