Дерево наследования уже трещит по швам? Не понятно, куда же «прикручивать» очередную фичу? Печально осознавать, но такие проблемы появляются даже если это далеко не первый наш проект.

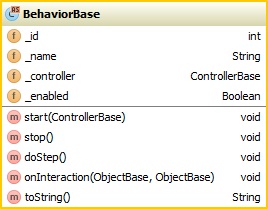
Одно из решений – концепция поведений. Все меняется, когда мы начинаем думать о реализации какого-либо свойства самого по себе а не в рамках определенного класса.

Например «разрушаемость». Разрушаемыми могут быть машины, дома, люди, мебель, да и вообще любые объекты игрового мира. Так почему бы не думать об этом как о поведении, присущем или не присущем объекту.

Тогда класс реализующий объект игрового мира будет всего один. А поведений множество. Своеобразный конструктор, из которого можно собрать все, что угодно.

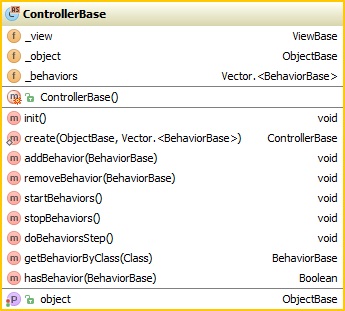
Конкретизируем на примере одного из моих текущих проектов. В общем имеем ObjectBase – основной и единственный класс игрового объекта. Контроллер, управляющий этим объектом, но на деле только реализующий поддержку поведений. Непосредственно поведения. И игровой мир, в данном случае просто поле, куда добавляются контроллеры всех объектов.

Но обо всем по-подробнее.



Базовый класс поведения. Как видим, мы можем запустить и остановить поведение. doStep() – это функция, которая вызывается в основном цикле приложения. Используется для реализации непрерывного поведения, такого как движение, например. onInteraction() - для реализации поведения, связанного с событиями. Один из примеров такого события – столкновение объектов.

Далее контроллер:

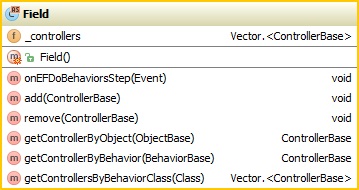


ControllerBase, класс который тоже в идеале должен быть один (т.е. никаких CarConroller и BuildingController).

Содержит в себе объект вида \_view, модели \_object, и список поведений \_behaviors. В него то мы и будем добавлять те самые BehaviorBase. Заслуживает внимания doBehaviorsStep() - он исполняет doStep() для всех поведений данного объекта.

Так же ControllerBase содержит ряд методов, для управления поведениями. Надеюсь, назначение их понятно из названий.

И непосредственно игровой мир:



В \_controllers мы добавляем контроллеры объектов (ControllerBase), напичканные поведениями. Основной момент – главный цикл приложения. onEFDoBehaviorsStep() выполняет doBehaviorsStep() для всех контроллеров объектов.

Здесь главный цикл намеренно назван onEFDoBehaviorsStep(), но совершенно не обязательно привязывать его к Event.ENTER\_FRAME. Возможно выполнять его каждый кадр, учитывая или не учитывая временной промежуток от предыдущего кадра. Возможно повесить его на setInterval().

В заключение пример, как выглядит в таком случае создание игрового объекта:

var car:ObjectBase = ObjectBase.create(/\*свойства типа позиции, материала, формы\*/ );

var carController: ControllerBase = ControllerBase.create(car, new <BehaviorBase>[

new UserControlBehavior (),

new MoveBehavior(),

new TrapBehavior(),

new BoostBehavior(),

new ShootBehavior(),

new DeathBehavior(),

new StatDisplayBehavior()]);

\_field.add(carController);

Объект, управляемый игроком, умеющий передвигаться, ставить ловушки, ускоряться, стрелять, умирать и о котором отображается статистика.

Для начала все. Во [второй части](sex:male) рассмотрим конкретику на примере MoveBehavior (поведения движения) и затронем огранизацию управления объектом пользователем и искусственным интеллектом(суть бот).