MVC:
Ursprunglig: View tog emot användarinput, View har access till controller-objektet och vice versa via komposition. CarController hanterade ytterligare funktionalitet utöver att bara hantera extern input från användare såsom instansiering av objekten i världen, samt hantering av en timer. CarController innehöll också en main metod och således skedde körning av programmet genom CarController. Då modellen inte innehöll bildinformation så fick drawpanel ansvara för tilldelning av bilder till de olika objekten, vilket inte uppfyller kriteriet om att View ska vara dum.

Ny: Controller håller inte längre ett view-objekt, view-objektet har dock tillgång till Controller-objektet via komposition. Controller modifierar nu bara modellen utifrån anrop från view som tar emot användarinput. Timer och instansiering av “världs”-objekten har lyfts ut till en ny klass som heter “Simulation”. DrawPanel är inte längre beroende av flera olika objekt i modellen utan endast toppskiktet i form av ett Interface.

Brister: En brist är att Controller och view lagras i fält i Simulation, vi kör alltså komposition istället för usage dependency. Objekt i världen bör egentligen instansieras i modellen, inte i Controller. Moveallcars och Collision-metoderna bör inte ligga i Controller utan snarare i modellen enligt MVC.

Åtgärder:
Observer: Vi använder oss av observers genom JPanel (Actionlisteners). Vi behöver då inte veta vad användaren faktiskt gör utan enbart när användaren genomför specifika actions. Hade kunnat användas i DrawPanel (antar att DrawPanel är en extension av view) så att DrawPanel lyssnar efter förändringar rörande vilka objekt som för tillfället finns i modellen.

Factory Method:
Vi använder oss inte av factory method i nuläget. Hade kunnat användas för att exempelvis skapa en bil samt minska coupling mellan simulation och modellen. Detta bör vi eventuellt göra.

State:
Vi använder oss inte av State i nuläget. Vi ser inte heller något större incitament till att vi skulle använda oss av State. Det hade eventuellt kunnat användas för att hantera ramp-funktionaliteten (att ha rampen uppe eller nere kan vara två olika states).

Composite:
Vi använder oss av Composite på flera ställen i programmet. Till exempel så används Composite för att se till att DrawPanel kan hantera olika typer av objekt på samma sätt. Composite används också för att se till att Loader bara kan kopplas till objekt som ska kunna lastas. Vi ser inte något incitament för att nyttja Composite pattern mer i programmet.