Flyttade Hanteringen av Drag and drop till DiceApp-klassen: I den ursprungliga koden hanterades drag- and drop funktionaliteten i Main-klassen. Jag flyttade denna funktionalitet till DiceApp-prototypen för att bättre följa OOP-principerna. Detta gjorde så att DiceApp själv hade kontroll över sin egna instans windowWrappers (html Div'et för tärningsapplikationsfönstret) positionering.

Bytte från drag and drop till mousemove, mouseup och mousedown: Istället för att använda HTML5's drag and drop API, som kan vara begränsande och orsakade problem i mitt fall, gick jag över till en mer manuell metod för att hantera mus-händelser och ge bättre kontroll över elementets rörelse genom att använda mig av "mousemove", "mousedown", "mouseup".

Justering av Z-Index Hantering: Den ursprungliga koden itererade genom alla fönsterelement och satte deras z-index. Och hanterades i Main klassen. Detta ansåg jag var dåligt då det innebar att en funktionalitet relaterad till DiceApp hanterades i Main. Men eftersom jag ville att all funktionalitet som berör tärningsapplikationsfönstret så flyttade jag justeringen av Z-indexet till DiceApp. Detta kom dock med nackdelen att DiceApp inte kunde veta vilket Z-index utomstående DiceApp instanser hade. En lösning jag började utreda var om jag i Main.js kunde hålla reda på alla DiceApp instanser som hade skapats och genom dom se vilka Z-index alla DiceApp instanser hade och sedan i DiceApp anropa en metod i Main för att justera så att aktuell DiceApp instans skulle få +1 Z-index än vad de andra instanserna hade. Men detta skulle innebära att DiceApp var tvungen att anropa metoder i Main.js trots att jag vill att varje klass endast ska kunna manipulera egenskaper i sina egna instanser. Som lösning använde jag mig istället av "new Date().getTime()/1000" för att vid varje justering av Z-indexet vara säker på att föregående Z-index som blivit satt inte kan vara högre än den Z-index värdet som sätts. Detta innebär att Z-indexet kan hanteras själv av DiceApp utan att behöva kommunicera med andra klasser eller att en annan klass ska manipulera något som bör hanteras av DiceApp själv.

Ändring av Dice.js: Eftersom Dice menar att endast hantera en instans av en enskild tärning ändrades Dice klassens metoder till "roll" och "destroy". Innan användes "add" och "roll" metoderna vilket dels stred mot sättet de olika klasserna bör ha hanterats. I "add" utfördes DOM manipulationer trots att jag bestämt att det endast var DiceApp som skulle hantera DOM manipulationer som sker i ett tärningsapplikationsfönster. "Roll" metoden var också felaktig då den hanterade funktionalitet för att ändra flertalet tärningars värdes trots att Dice endast ska representera en enskild tärning. "Add" och "Roll" slogs istället ihop till samma metod kallas "roll" som nu endast slumpar ett tall mellan 1-6 och sätter sedan Dice's egenskap value till detta värde. Metoden "Destroy" lades till för att ta bort instansen så att garbage collectorn tar bort instanser av Dice som inte längre används.