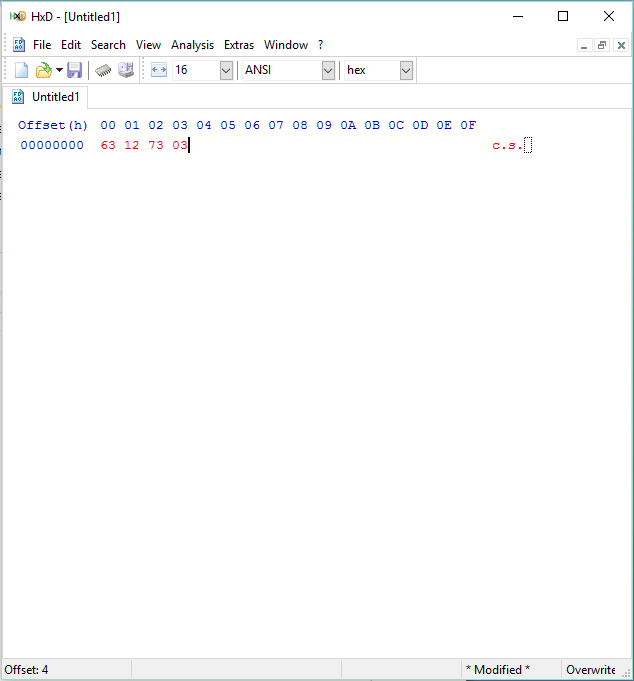
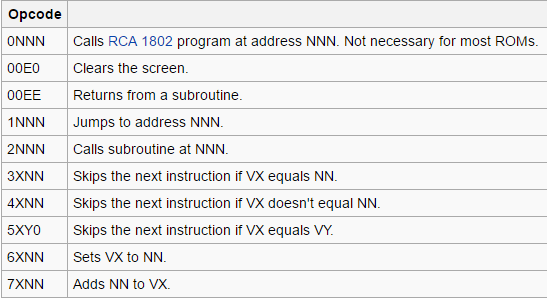
**Chip 8 Testing**

**Testing av opkoder**

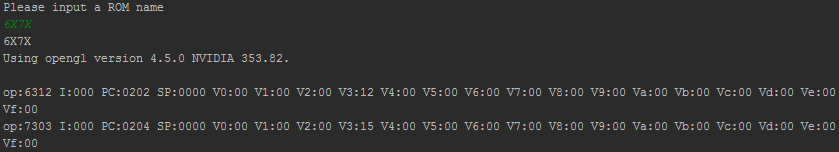
For å teste alle opkodene våre utledet vi individuelle tester for hver eneste en og til slutt laget tester som testet dem i grupper. For å gjøre dette opkodene lagde vi test ROMs i en hexadecimal editor. Et slikt redigeringsprogram lar oss skrive inn heksaadesimal input som vi da kan lagre som en ROM fil og laste inn i ROM loaderen vår. Når loaderen kjører filen så leser den gjennom den heksadesimale inputen som programmet da tolker som forskjellige opkoder den skal gjøre noe med.   
  


*(Figur X.1) HxD Hovedmeny. HxD er et gratisprogram utviklet av* [*MH-Nexus*](https://mh-nexus.de/en)

La oss nå prøve oss med litt testing. Vi kan ta hex-verdiene fra figur X.X og kjøre dem gjennom ROM loaderen vår. Det første vi vil vite er jo hvilken opkoder som kjører og hva de gjør. Opkoder er identifisert i grupper av 4. Dette vil si at ”63 12” er en operasjon, og 73 03 en annen. For å finne ut hvilken opkode som blir kjørt og hva den gjør så tar vi en titt på figuren under.

  
*(Figur X.2) En liten del av opkode tabellen fra* [*Chip8*](https://en.wikipedia.org/wiki/CHIP-8) *sin Wikipedia side.*

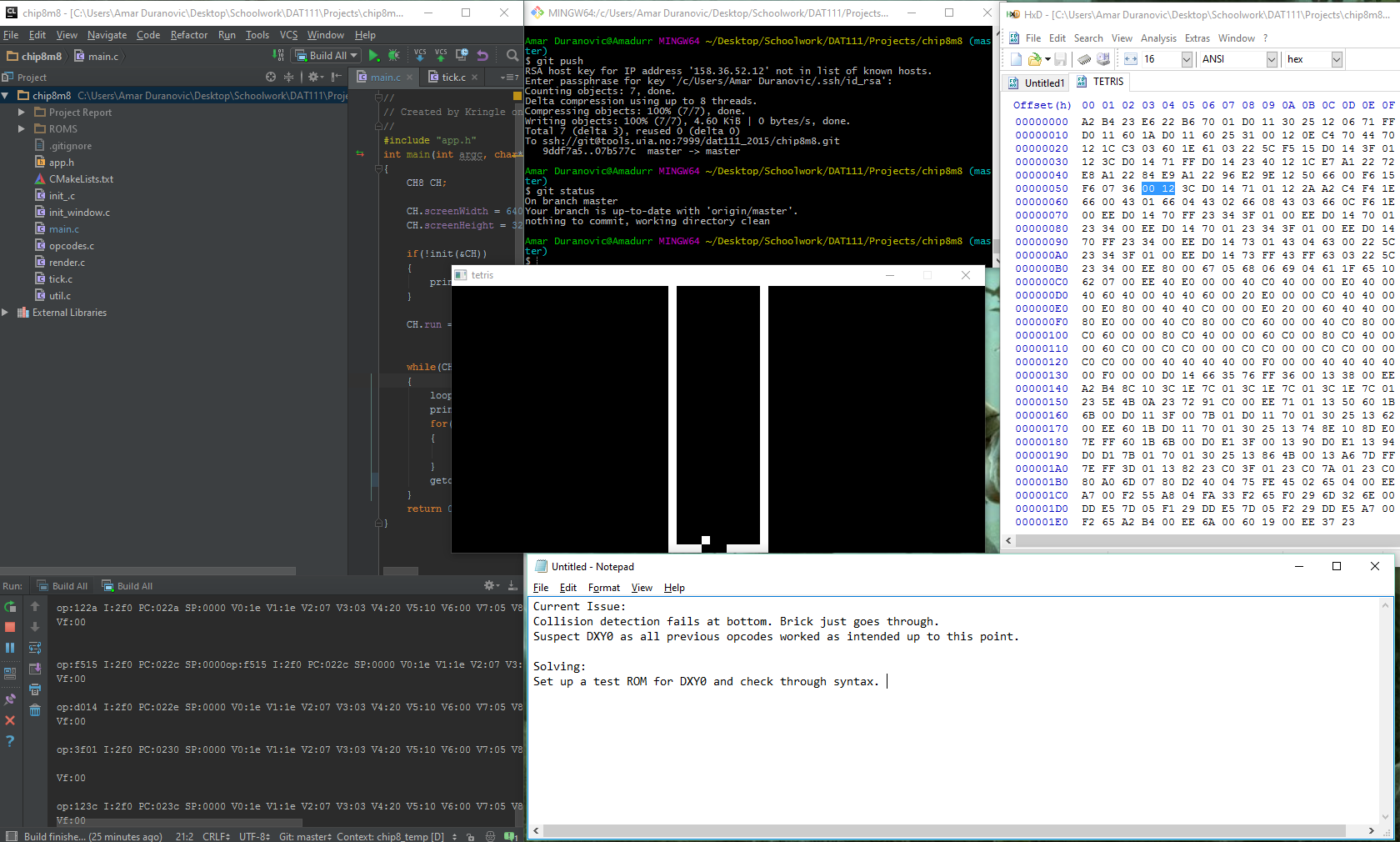
Det første sifferet identifiserer hvilken opkode som skal kjøres. I det første tilfellet ’6’ så sier det oss at det er opkode 6XNN som kjøres. For de 3 neste siffrene trenger vi bare følge beskrivelsen til opkoden i tabellen. I dette tilfellet blir desimalverdien 18 (12 er den heksadesimale verdien, som da gir oss 18 desimalt) lagt inn i register nummer 3. Etter denne operasjonen er utført blir opkode 7XNN utført. Den legger verdien 3 til register V3. Dette betyr at det nå burde ligge en total av 21 på register V3. Det sjekker vi i konsollen:



*(Figur X.3) Konsollen til programmet.*

Ved første opkode ser vi at den kjører 6312 som planlagt og setter 12 (Heksadesimalt) inn i V3. Da vet vi at den fungerer som den skal i dette tilfellet. Etterpå ser vi at op:7303 kjøres og den legger til 3 i V3, som gir en total av 15 (21 desimalt). Slik ser vi at opkode 6XNN og 7XNN fungerer slik som de skal i dette tilfellet.

**Feilsøking**

****(Figur X.4) Jakten på bugs.

Når man utvikler noe så er man så og si garantert for å komme borti feil. I figur X.4 ser vi en ganske morsom feil vi fikk når vi prøvde å kjøre tetris. Den tegna opp banen og slapp brikken uten problemer, men når brikken kom ned til bunnen gikk den bare gjennom banen å stoppet programmet. Etter en god latter satte vi oss ned og analyserte problemet. Vi steppet programmet frem til hvor feilen skjedde, fant ut hvilke opkoder som ble brukt og så sjekket dem individuelt. Vi lærte at når feil oppstår så er det viktig og sette seg ned å få et perspektiv over problemet. Når man har en ide om hva som kan være feil setter man i gang med prosessen som er nødvendig for å fikse disse feilene.   
  
Etter vi gikk gjennom syntaxen vår å laget en ekstra test fant vi ut at vi hadde gjort en feil på opkode DXY0. Etter en liten forandring kjørte vi tetris på ny og da fungerte alt slik som det skulle!