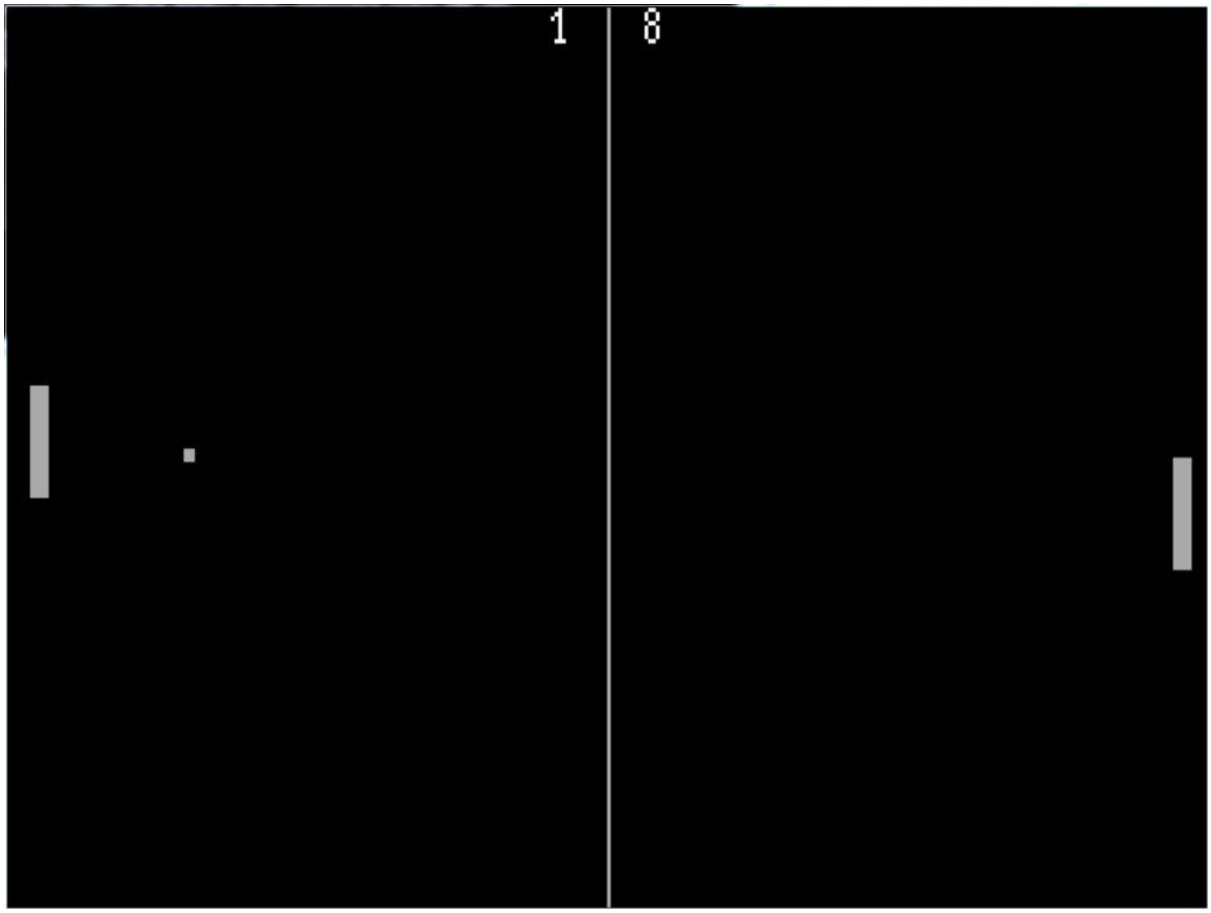
Computersystemen - project

Vrije Universiteit Brussel – Computerwetenschappen

academiejaar 2016 - 2017



Senne Deproost
senne.deproost@vub.be
#0528402
•
Seppe Seghers
seppe.seghers@vub.be
#0524686

Inhoud

1. Algemene beschrijving

Dit is een document dat het project computersystemen voor 2de BA Computerwetenschappen beschrijft voor academiejaar 2016-2107. De opdracht was om een programma van redelijke omvang te schrijven in MASM Assembly volgens de Intel x86 syntax. Voor het voltooien van het programma werd er gebruik gemaakt van teksteditors Notepad++ en Atom alsook source control software git en de DOSBox omgeving om de software te testen.

Onze bedoeling was om voor deze opdracht het spel Pong te recreëren, waarbij er een soort van rudimentaire 2D tafeltennis gespeeld wordt tussen twee spelers. In dit geval is dit tussen de gebruiker en een computergestuurde tegenspeler. Het programma dat wij hebben geschreven laat spelers toe om het spel te spelen tegen een “A.I.”, waarbij een score wordt bijgehouden die van 0 tot 9 gaat. De eerste persoon die 10 punten scoort heeft gewonnen. In dit geval stopt het spel en wordt er aangeduid wie de winnaar is aan de hand van symbolen die op het scherm verschijnen (een kroontje en een doodskop). Deze nummers en tekeningen zijn opgeslagen in het DATASEG als een array van bits, waarbij 0 een zwart vakje voorstelt en 1 een wit vakje.

Het hoofdonderdeel van onze code is de gameloop. Van hieruit wordt alles opgeroepen wat moet worden uitgevoerd. Nadien wordt er gesprongen naar het begin van de loop en worden alle procedures opnieuw uitgevoerd. Elke iteratie van de gameloop worden er calls gemaakt om te zien wat de gebruiker heeft ingedrukt, om de objecten te bewegen, om te zien of de bal ergens tegenaan botst, om alles te tekenen, en om te testen of het einde van het spel is bereikt. In dit laatste onderdeel kijkt het programma of er één van de twee spelers een score heeft die gelijk is aan 10. Als dit zo is wordt er uit de loop gegaan en wordt er naar de end_prog loop gesprongen. Hier wordt enkel nog de input van de gebruiker gecontroleerd om het spel af te sluiten.

Alle input van de gebruiker gebeurt via het toetsenbord. Er zijn 5 toetsen die gebruikt worden in het spel: Escape, ←, ↑, ↓ en →, elk met hun eigen nut. De pijltjestoetsen ↑ en ↓ dienen voor de bediening van het pallet dat de gebruiker bestuurt (linkerpallet), respectievelijk naar boven bewegen, en naar beneden bewegen. De Escape toets sluit het spel en keert terug naar de DOSBox applicatie. De pijltjestoetsen ← en → dienen voor het bepalen van de snelheid van het rechterpallet, dat bestuurd wordt door de computer. Al deze code hiervoor staat geschreven in de handleUserInput functie, die in de keyboard BIOS zoekt of deze keys worden aangehaald. Deze functie wordt bij elk doorlopen van de gameloop opgeroepen, ook als het spel in het eindstadium is.

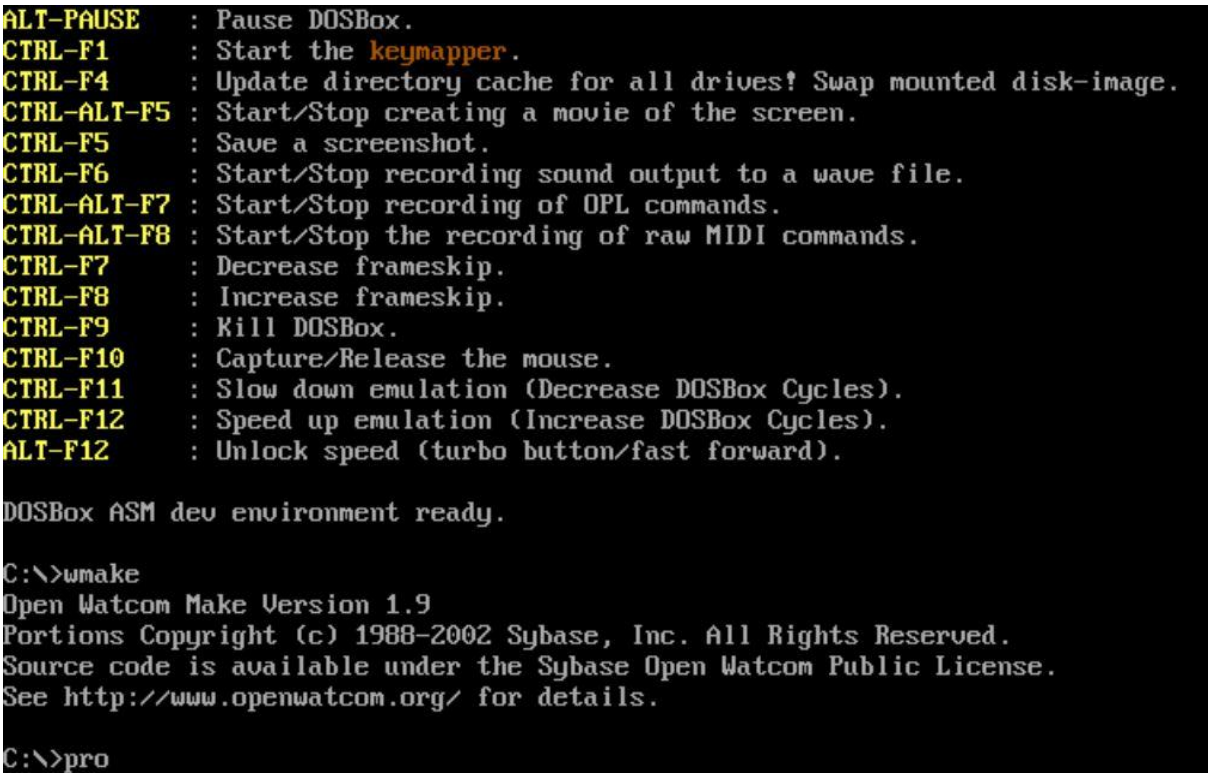
We gebruiken een simpele maar effectieve A.I. om het pallet van de tegenspeler te besturen. Deze is te verslagen aan de hand van de juiste trucjes, maar zorgt toch nog voor verbazingwekkend spannende spelmomenten.

2. User guide

Benodigdheden:
• ASMBox • makefile en .asm bestand

Voor het eerst opstarten:
1. De ASMBox.zip-file extracten 2. Zoek op en voer ASMBox.bat (Windows) of ASMBox.app (Mac OS X) uit. 3. Er zou nu een nieuwe folder met de naam ‘c_disk’ moeten zijn. 4. Plaats makefile en .asm bestand in c_disk. 5. Open ASMBox.bat of ASMBox.app. 6. Voer “wmake” in in het scherm dat net is geopend. 7. Voer nu “PRO” in om het spel voor een eerste keer uit te voeren.

Normaal stappenplan:
1. Open ASMBox.bat of ASMBox.app. 2. Voer “PRO” in om het spel te spelen.



Bediening:
• Pijltjes toetsen om de hoogte van het pallet van de speler te bedienen (↑ = naar boven, ↓ = naar beneden) • 'Escape' toets sluit het spel.

3. Data model

In onze code gebruiken we twee soorten variabelen, globale en lokale. Globale variabelen staan onderaan in DATASEG geïmplementeerd met hun beginwaarde, hierdoor kan elke functie deze gemakkelijk aanpassen. Dit wordt dus gebruikt door variabelen die op meerdere plaatsen worden geraadpleegd, of variabelen die over verschillende iteraties van de gameloop moeten meegaan. Lokale variabelen zijn variabelen die enkel op een bepaalde plaats op een bepaald tijdstip worden gebruikt, en nadien of op een andere plek niet meer beschikbaar zijn. DATASEG gebruiken we ook om variabelen in op te slaan die zo maar op één plek te moeten worden veranderd om effect te hebben op heel het programma. Zo houden we bijvoorbeeld ballWidth bij, voor het geval we ooit zouden beslissen om een bredere bal te gebruiken, zonder dit op iedere lijn die dit gebruikt te moeten aanpassen. Dit is nog wel niet perfect, op sommige plaatsen worden nog vaste waarden gebruikt voor variabelen die wel in DATASEG staan om deze reden.

Het schrijven naar grafische modus doen we aan de hand van een ScreenBuffer, dit is een array die initieel gevuld is met nullen, waar we de huidige objecten die moeten worden getekend in opschrijven, en dan in 1 keer naar de videomodus te sturen. Dit doen we om een probleem tegen te gaan waar we flikkeringen kregen op het scherm doordat elk object de grafische modus individueel aansprak, ook als die eventueel nog bezig was met het tekenen van vorige objecten.

4. Problemen & Oplossingen

4.1. Bal bewegen

Het eerste probleem dat we tegenkwamen was de bal die moest bewegen over het scherm op een manier die simpel, logisch, en makkelijk te veranderen was. Hiervoor hebben we gekozen om in het DATASEG twee variabelen bij te houden, namelijk ball_angle en ball_direction, die respectievelijk bijhouden wat de huidige hoek (richtingscoëfficiënt) en richting t.o.v. de x-as is. Op deze manier is dus ook een move_Ball functie ingevoerd die simpelweg de huidige coördinaten van de bal opvraagt, bij de x-coördinaat de angle aftrekt en bij de y-coördinaat de direction optelt. Zo kan er dus snel de volgende positie berekend worden terwijl andere delen van de code deze variabelen ook gemakkelijk kunnen aanpassen op efficiënte en logische manieren. Bij de collision met de bovenste rand bijvoorbeeld is het heel simpel om gewoon de ball_angle te vermenigvuldigen met -1 voor deze botsing, iets wat anders meer rekenwerk zou vereisen.

```
575   mov ax, [ball_x_pos]
      add ax, [ball_direction]
```

```
      mov [ball_x_pos], ax

      mov ax, [ball_y_pos]
      mov dx, [ball_angle]
      neg dx
      add ax, dx
583   mov [ball_y_pos], ax
```

4.2. Flikkerend scherm

Een ander, toch wel enerverend probleem waar we op stootten, was de manier waarop de videomodus de beelden afdrukte die wij stuurden. Van frame tot frame kon het voorkomen dat er al een volgende iteratie van de gameloop bezig was voor de vorige beelden op het scherm waren getekend. Dit is op zich vrij normaal op zo'n hoge frequenties, maar omdat er geen buffer was betekende dit dat het tekenen halverwege werd stopgezet wat resulteerde in constante flikkeringen op het scherm, die niet echt aangenaam waren.
Dit is opgelost door een array te schrijven die als buffer werkt voor alles wat naar de videomodus wordt gestuurd, waardoor de videomodus eerst het huidige beeld kan afwerken alvorens het volgende te beginnen tekenen.

```
1064  ; VIDEO BUFFER

      mov esi, offset offset_screenBuffer
                  ; points to a "db 64000 dup" array
      mov edi, [startaddr] ; the video memory
      mov ecx, 64000 / 4 ; 320 * 200 , but copy groups four bytes
1069  rep movsd ; moves a dword and updates ecx , e s i and edi
```

Deze array bevat dus initieel allemaal nullen.

4.3. Tegenspeler

We wilden dit spel single player maken, net zoals de originele Pong, en daarvoor was er dus een A.I. nodig als tegenspeler. Dit heeft voor wat problemen gezorgd omdat er direct twee manieren in ons opkwamen om dit te realiseren, namelijk een simpele, maar brute manier, en een iets geavanceerdere maar elegantere manier. Eerst waren we van plan om gewoon bij elke iteratie van de gameloop movePaddle2 de huidige y-coördinaat van de bal mee te geven, waardoor het rechter pallet constant de bal zou volgen. Dit zou als problemen hebben dat er geen enkel manier is om te scoren aan de rechterkant, en ook dat het pallet plotseling van positie zou verspringen als de bal reset na een punt. Hierdoor hebben we beslist om het pallet altijd het verschil met de bal te laten berekenen, en dan de y-coördinaat van het pallet aan te passen met een bepaalde variabele paddle2_speed, in de juiste richting.

Een probleem dat zich voordoet bij paddle2 is dat de padle flickert. Dit wordt groter naarmate men de waarde voor paddle2_speed aanpast. Een oorzaak voor deze bug werd niet gevonden, maar het zichtbare effect ervan is klein voor waarden < 3.

```
664   mov ax, [paddle2_y_pos]
      mov bx, [ball_y_pos]

      cmp ax, bx
      jge @@moveUp
      @@moveDown:
            add ax, [paddle2_speed]
```

```
            jmp @@end
      @@moveUp:
            mov bx, [paddle2_speed]
            neg bx
            add ax, bx
            jmp @@end


      @@decrement:
            sub ax, [paddle2_speed]
            jmp @@end


      @@increment:
            add ax, [paddle2_speed]
            jmp @@end

      @@end:
695         mov [paddle2_y_pos], ax
```

4.4. Paddles buiten scherm

Doordat de paddles kunnen bewegen met een stapgrootte groter dan 1, was het niet voldoende om aan de onder- en bovenrand een compare te laten lopen die checkt of de y-coördinaat van de pallets gelijk was aan 0 of 200. Ook wisten we niet echt hoe we het pallet moesten tegenhouden aan deze limieten, er was de mogelijkheid om de verplaatsing gewoon niet te laten doorgaan of bijvoorbeeld het pallet een soort van cooldown geven waarin geen beweging mogelijk was. Uiteindelijk hebben we besloten om de verplaatsing tegen te gaan, en direct en ter plaatse ook een beweging in de andere richting uit te voeren. Hierdoor lijkt het op het scherm dat het pallet de muur niet door kan en terugbotst, en behoudt de gebruiker de volledige controle over de beweging.

```
667   ; Controle of de paddle niet buiten scherm dreigt te gaan.
      cmp ax, 0
      JLE @@increment
      cmp ax, 175
671    JGE @@decrement
```

Hier is de waarde 0 gekozen als bovenlimiet, en 175 als onderlimiet, dit omdat het scherm 200 hoog is, en het pallet 25 heeft als hoogte. Het punt dat wij beschouwen als pallet is eigenlijk de linkerbovenhoek van de figuur.

4.5. Score

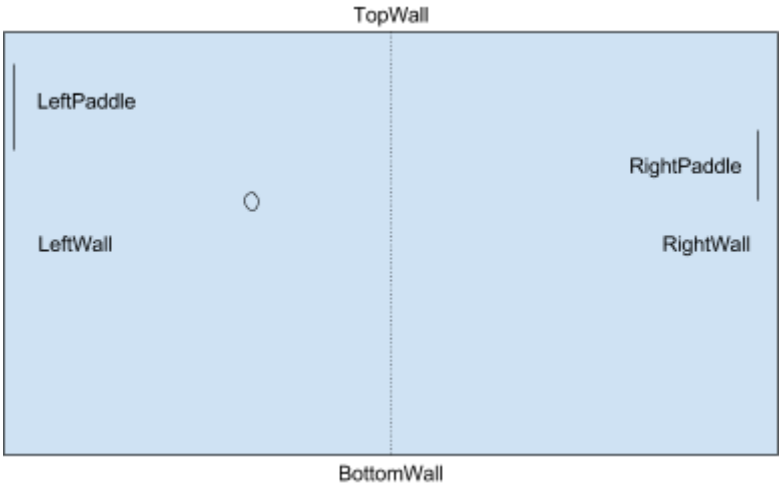
Nog een belangrijke stap in ons werkend spel is een eind conditie waar het spel kan stoppen als een bepaalde toestand is bereikt. Met deze reden hebben we een scorebord ingevoerd, waarop de huidige puntenstand wordt weergegeven. Indien één van de twee scores 10 behaalt eindigt het spel en gaat het spel in en nieuwe loop waar aan de hand van symbolen de winnaar wordt aangeduid en het spel op ‘pauze’ staat door een nieuwe loop waar enkel de input van de gebruiker wordt opgevraagd.

Om deze score te laten werken zouden we nummers moeten kunnen doen verschijnen op het scherm, en deze gemakkelijk op dezelfde plaats van nummer laten veranderen.
Aan de hand van deze eisen hebben we besloten om 10 nummers en 2 symbolen bij te houden in DATASEG in een array van bits. Er worden twee globale variabelen score1 en score2 bijgehouden, die door middel van drawScore de juiste sprite van een nummer of symbool op het scherm plaatst.

Het is enkel mogelijk om sprites bovenaan het scherm te tekenen. Dit komt omdat de y coördinaat binnen de procedure drawSprite vastgezet is op 0. Wanneer er geprobeerd werd om een score te tekenen op y = 0 en op een bepaalde x, was de y coördinaat gelijk aan de waarde van de score. Voor dit probleem werd er niet meteen een oplossing gevonden, vandaar de vaste waarde voor y.

4.6. Collision

Nog een groot probleem was dat de bal en de pallets wel konden bewegen, maar geen interactie met elkaar hadden. Hiervoor hebben we de checkCollision functie geschreven.
De collision is de botsing die een bal maakt wanneer hij botst met 1 van de 6 randen en pallets. Deze heten in de code LeftWall, RightWall, LeftPaddle, RightPaddle, TopWall en BottomWall.



Collision is een functie die bij elke iteratie van gameloop wordt opgeroepen om te berekenen of er een botsing is, en wat er hierdoor gebeurt.

Dit hebben we moeten implementeren op een manier waar 6 verschillende functies worden opgeroepen die elk instaan voor hun eigen object waartegen de bal kan botsen. Elke functie berekent dan zelf of er een botsing plaatsvindt, en voert de nodige effecten uit. Bij Left- en RightWall verhoogt één van de scores als de bal raakt en start de bal opnieuw in het midden. Top- en BottomWall vermenigvuldigen de BallAngle met -1, maar laten de BallDirection zo, waardoor de bal schijnbaar botst. De paddles zijn wat ingewikkelder en vereisen grotere functies, omdat elk deel van de paddle de BallAngle met een specifieke waarde verhoogt of verkleint.

De bal kan soms vastlopen aan een paddle. Deze moet dan voortbewogen worden om de bal los te krijgen. Dit kan bij paddle2 gedaan worden door de linkerpijltjestoets zodanig in te drukken dat paddle2_speed negatief wordt en de paddle zich ervan losrukt.

4.7. Spel balanceren

Een van de laatste problemen die we tegenkwamen was dat het spel niet gebalanceerd was, doordat de A.I. overal zonder problemen bij was. De oplossing die we hiervoor hebben gevonden is niet de beste of de elegantste, maar doet voorlopig wel zijn taak. We wilden een soort van mechanisme implementeren waardoor de gebruiker zelf de snelheid van het pallet van de A.I. kan besturen.
Doordat de rest van het spel alleen de pijltjes naar boven of beneden gebruikt, waren er nog twee pijltjestoetsen over die we hiervoor kunnen gebruiken.
Een duw op het pijltje naar links zorgt ervoor dat de snelheid met 1 vermindert, rechts verhoogt de snelheid met 1. De snelheid zelf kan niet negatief worden, aangezien dit voor problemen zorgde met de A.I., maar een lagere snelheid zorgt er wel voor dat het spel speelbaar en spannend wordt. Het grote probleem met deze maatregel is dat de eindgebruiker dit kan misbruiken om de paddle van de tegenstander bijvoorbeeld gewoon stop te zetten.

4.8. Video buffer

Normaal gezien zouden alle elementen die getekend moeten worden op het scherm eerst naar de videobuffer verstuurd moeten worden. Helaas is dit enkel het geval voor cijfers van het scorebord en het doodskop en kroon symbool. De rest, ball, midline en de paddles, worden meteen op het scherm getekend via int 10h.

5. Resultaat

Het behaalde resultaat is een speelbare variant van PONG met scorebord en verschillende moeilijkheidsgraden. De vooropgestelde doelen zijn gehaald, ondanks gemaakte compromissen, en het project is op tijd ingediend.