VRIJE
UNIVERSITEIT
BRUSSEL

Project Computersystemen

Breakout

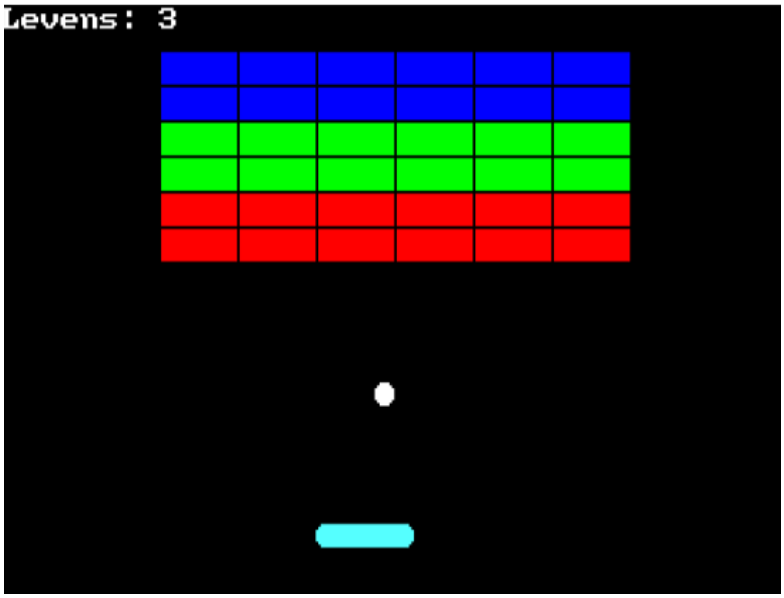
Groep G19

Maxim Brabants & Alexandre Fernandes Medeiros

December 26 2021

Academiejaar 2021-2022

Computerwetenschappen



Figuur 1: Voorbeeld Breakout

1 Introductie

Voor ons project hebben we ons gericht op het befaamde retrospel dat bekend staat onder de naam: ‘Breakout’. Op basis van dit spel zijn er ontelbaar andere spellen gemaakt, maar wij kozen er uiteraard voor om de meest originele versie te implementeren (weliswaar zonder power-ups)

Het speelveld van het spel in kwestie kan in feite onderverdeeld worden in drie componenten. Een component bovenaan waarin zich een grid van stenen bevindt met per twee rijen een andere kleur. Onderaan heb je het gedeelte waar een peddel zich voortbeweegt in horizontale richting en ten slotte heb je nog rondom de stenen en boven de peddel de ruimte waarin een bal het meeste van de tijd gaat bewegen.

De bedoeling van ons spel is eigenlijk heel simpel en rechttoe rechtaan. Aan het begin van het spel start de peddel samen met de bal onderaan het scherm waarbij aanvankelijk de bal nog vastzit aan de peddel zodat de speler initieel nog kan kiezen vanop welke positie hij de bal afvuurt naar het raster van stenen. Als de bal is afgevuurd, beweegt deze zich voort over het scherm, richting de stenen, tot op het moment dat de bal één van de stenen (of schermgrenzen) raakt. In dat geval verdwijnt de steen in kwestie en vervolgens kaatst de bal af en beweegt deze zich dan verder voort. Ook kan de bal gekaatst worden door de speler zelf als deze in staat is de peddel recht onder de bal te krijgen. De speler verliest een leven wanneer de bal onder de peddel het scherm verlaat of anders gezegd: wanneer de speler de bal niet heeft kunnen kaatsen met de peddel en de bal dus gemist heeft.

2 Korte manual

Ons programma kan uitgevoerd worden door de 'MAIN'-executable te runnen via ASM-Box. Dat bestand bevindt zich in de spelmap. Bij aanvang van het spel is er geen menu-interface voorzien dus het spel start onmiddellijk met alle componenten zichtbaar zoals eerder besproken.

De speler kan horizontaal bewegen met de pijltjestoetsen waarbij de bal vastzit aan de peddel. Indien de speler de bal wenst af te vuren richting de stenen, dient hij/zij eenmalig op de spatiebalk te drukken. Vanaf dan kan de speler verder horizontaal bewegen onderaan het scherm eveneens door gebruik te maken van de pijltjestoetsen. Verder zijn het resterende aantal levens zichtbaar in de linkerbovenhoek. Als de speler de bal mist met de peddel dan wordt deze met één verminderd. Daarna keren de bal en de peddel terug naar hun startposities (met de bal vast aan de peddel).

3 Programmadesign

Onder deze sectie zullen we bespreken welke onderdelen we hebben geïmplementeerd en op welke manier dit werd gedaan, etc. Hieronder een overzicht van de te bespreken onderdelen:

- Onze programmastructuur (spellus,…)
- De voorstelling van zichtbare objecten in het speelveld.
- Het tekenen van onze sprites op het scherm in videomode 13h.
- De beweging van bepaalde objecten.
- Raken van stenen met bal

3.1 Programmastructuur

Eerst en vooral zullen we bespreken hoe onze code precies in elkaar zit en hoe deze programmatorisch is opgebouwd. Dit houdt ook in welke structurele keuzes we hebben gemaakt om onze code een logische opbouw te geven.

Om te beginnen dachten we te opteren voor een aanpak die redelijk sterk overeenkomt met wat we bij het Space Invaders spel moesten doen of wat toch aangewezen was. Deze aanpak houdt hoofdzakelijk in: het zo goed mogelijk opsplitsen van de spellogica en de tekenlogica. We hebben weliswaar zo goed mogelijk geprobeerd om ons hieraan te houden.

Om dit te bereiken hebben we in feite twee belangrijke procedures voorzien die een grote rol spelen in ons programma. Eén procedure die de rol van het tekenen op zich neemt en de andere die alles wat te maken heeft met de spellogica afhandelt. In het entry point van ons programma bundelen we deze dan samen om zo tot een volwaardig spel te komen (weliswaar met enkele details maar daar wordt onder deze sectie niet verder op ingegaan). Hetgeen dat ons spel draaiende houdt, roept dan die twee onderdelen op zodat spel -en tekenlogica in wezen parallel kunnen opereren maar eigenlijk toch telkens één voor één worden opgeroepen. Verder splitsten we onze code nog op in enkele bestanden om zo de overzichtelijkheid ten goede te laten komen. (algemene procedures, globale constanten & structs, …)

3.2 Voorstelling objecten

Aanvankelijk zaten we met de vraag: “Hoe gaan we de objecten die zichtbaar zijn op het scherm, voorstellen in onze code?”. Die vraag was vrijwel onmiddellijk beantwoord door het feit dat we in C een heel handig concept hebben gezien dat zich definieert als ‘Data Structures’. Als gevolg hiervan hebben we drie zogenoemde ‘Data Structures’ (STRUCTS) gedefinieerd in onze code die de peddel, bal en een steen voorstelt in het spel.

Hier volgt een korte beschrijving van deze structuren:

- Voor onze bal (Ball) houden we zijn x -en y-coördinaat bij, of hij al dan niet actief is (aan de peddel vastzit of niet) en ten slotte zijn richting over de x -en y-as. Deze zijn allemaal voorgesteld in de vorm van één byte.
- De datastructuur voor de peddel (Paddle) bevat ook de x -en y-coördinaat samen met het aantal levens. Ook allemaal velden van één byte groot.
- De Steen (Stone) encapsuleren we slechts met één staat die aangeeft of hij ‘levend’ is of niet. Hoewel onze steenstructuur aanvankelijk omvangrijker was, hebben we ons toch beperkt tot maar één attribuut als het ware met de reden dat het overbodige elementen waren.

3.3 Tekenen sprites

Omdat de peddel, bal en de stenen allemaal een unieke vorm hebben, waren we genoodzaakt om te werken met sprites. Voor elk van deze hebben we byte-arrays bijgehouden in het datasegment (arrays met elk element ter grootte van één byte). Deze arrays hadden we wel nog niet onmiddellijk ter beschikking in het begin. Het was de bedoeling dat we eerst ontwerpen bedachten en deze dan tekenden in een tekenprogramma. Vervolgens moesten we een array zien te verkrijgen uit de image met de kleurenindex per pixel. Hieronder is een voorbeeld dat illustreert hoe zulke array eruitziet.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000010	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000020	0B	0B	0B	0B	0B	0B	00	00	00	0B	0B	0B	0B	0B	0B	0B
00000030	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000040	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	00
00000050	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000060	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000070	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000080	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000090	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000A0	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000B0	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000C0	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000D0	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000E0	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
000000F0	00	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000100	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000110	0B	0B	0B	0B	0B	0B	0B	00	00	00	0B	0B	0B	0B	0B	0B
00000120	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B
00000130	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	0B	00	00

Figuur 2: Voorbeeld byte-array peddel

Als we dan deze sprites op het scherm wilden krijgen, implementeerden we een procedure die op basis van een x -en y-coördinaat, de juiste offset van een sprite en zijn hoogte en breedte in pixels, deze sprite op de overeenkomstige positie op het scherm tekent. Dit is dan ook het deel waarbij we van een stel coördinaten over moeten schakelen naar een positie in pixels om de garantie op de scheiding in teken -en spellogica te waarborgen. Hiervoor hebben we een kleine abstractie gemaakt die een vakje op het scherm voorstelt namelijk een 'cel'. We laten zo'n cel overeenkomen met 4 pixels x 4 pixels.

Om te voorkomen dat we in het begin al onze stenen een x -en y-coördinaat moeten toekennen, wat duidelijk een O(n)-procedure is, hebben we in plaats daarvan voor elke steen zijn x -en y-coördinaat berekend zoals vooraf vermeld. Hierbij moesten we de stack gebruiken omwille van een tekort aan registers. De berekeningen worden hieronder getoond. De eerste macro stelt de x-coördinaat voor van de eerste steen (linksboven), de counter is de index van de huidige steen in de array, want we houden een array bij van steenstructuren. 'COLSTONES' stelt het aantal stenen voor in één rij. STONEWIDTHCELL is de breedte van een steen in term van cellen.

```
posx = STONESSTARTX + (counter%COLSTONES) * STONEWIDTHCELL
posy = STONESSTARTY + (counter/COLSTONES) * STONEHEIGHTCELL
```

Merk op dat we voor geen enkele steen de kleur bijhouden want dit zou ons wederom een pre-initialisatie kosten. In plaats daarvan zorgen we ervoor dat de byte-arrays voor de gekleurde stenen in de juiste volgorde in ons geheugen opgeslagen staan waardoor we als volgt te werk kunnen gaan: we doen een simpele berekening op basis van de hoeveelste steen dat we verwerken en het resultaat hiervan selecteert automatisch de juiste offset van de gekleurde steen. Deze berekening staat hieronder. De kleurenindex bekomen we door de huidige index in de array geheel te delen door het product van het aantal kolommen stenen in de raster en het aantal rijen van elke kleur. Dit geeft ons dan onze zes rijen stenen met telkens per twee rijen een andere kleur. ('grootte_sprite' = grootte van sprite in pixels).

```
(offset_eerste_sprite + kleur_index * grootte_sprite)
```

3.4 Beweging van objecten

Ervan uitgaande dat in elke iteratie van de spellus, al onze objecten worden hertekend, hoeven we niets meer te doen dan het object zijn corresponderende x -en/of y-waarden te manipuleren om de gewenste beweging van de sprite op het scherm waar te kunnen nemen. Elke keer voordat we een beweging doorvoeren op een object, checken we nog of deze niet de spelranden overschrijdt. Dit wordt gedaan in elke beweegprocedure die we in ons programma hebben staan. Voor de bal is dit een redelijk intensief proces, aangezien we niet enkel moeten checken of deze de spelranden raakt maar ook of deze de peddel of stenen raakt.

3.5 Raken van stenen

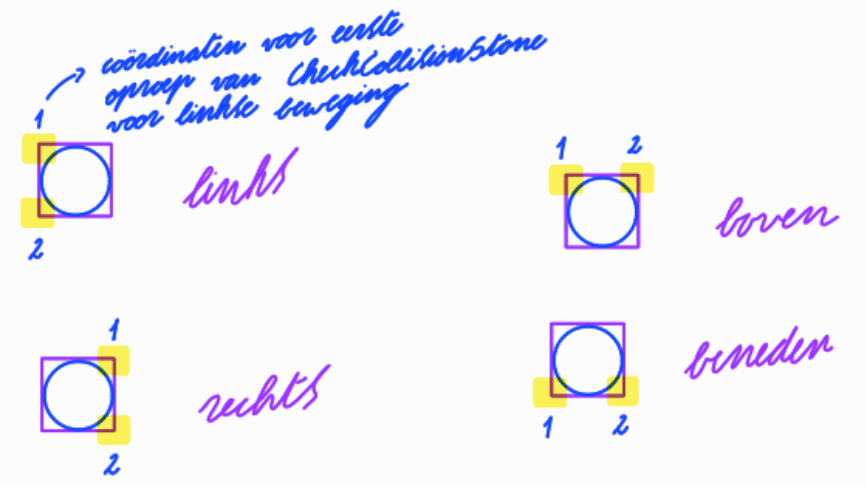
Om het overzicht een beetje te bewaren hebben we het procedé voor het raken van de bal met de stenen afgezonderd. Hierbij zijn 3 procedures van belang.

1. CheckCollisionStone: Gaat na of er overlapping is tussen de bal en één steen, indien de bal zich binnen de raster van stenen bevindt, maakt deze procedure gebruik van de procedure

"StoneAlive" om na te gaan of de steen op die positie nog "levend" is of niet. Om te weten als er echt een botsing plaatsvond.

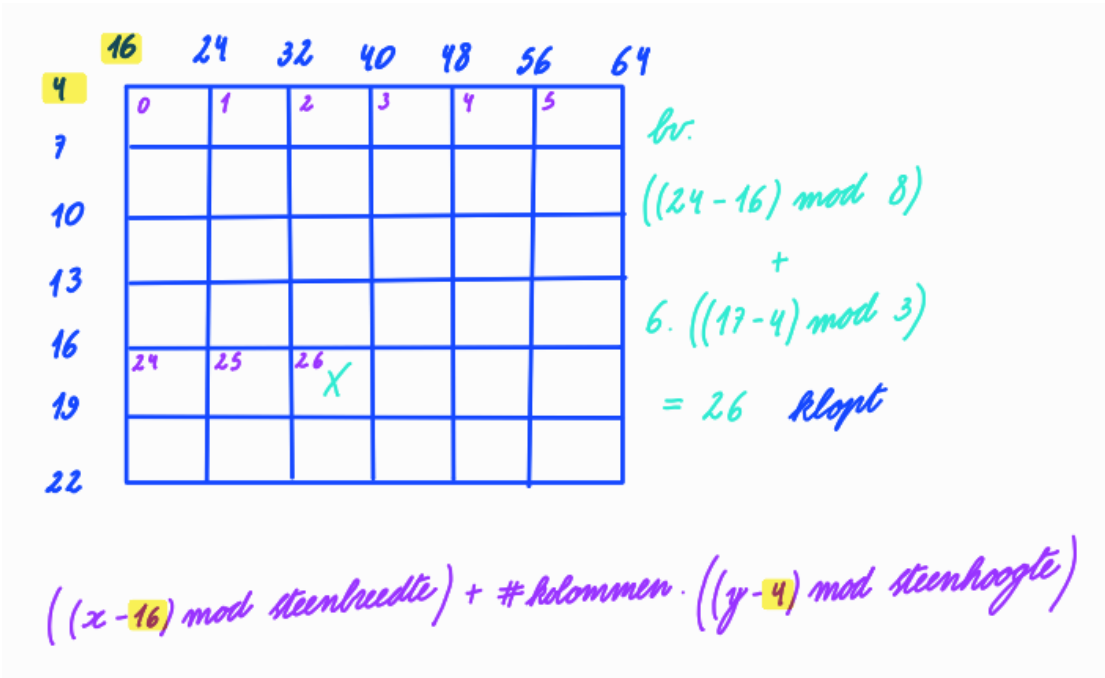
2. StoneAlive: Gaat a.d.h.v. van 2 coördinaten na welke steen zich daar bevindt/bevond en genereert een returnwaarde om te laten weten of de steen op die positie nog '"levend" is.
3. StonesAlive: Gaat na of alle stenen van de raster vernietigd werden en de speler dus heeft gewonnen.

Onderstaande figuur toont aan welke punten van de bal steeds in rekening worden gebracht bij het nagaan of de bal tegen een steen botst. Deze punten zijn natuurlijk afhankelijk van de bewegingsrichting. Als de bal bv. naar links beweegt, worden de twee linkse punten in rekening gebracht.



Figuur 3: punten in rekening gebracht

De volgende afbeelding toont aan hoe we aan onze formule zijn gekomen om a.d.h.v. twee coördinaten te bepalen welke steen hiermee overeenkomt. Deze formule wordt in de procedure "StoneAlive" gebruikt.



Figuur 4: formule uit "StoneAlive"

4 Enkele problemen

Hieronder volgt een lijst van problemen die we ondervonden hebben tijdens de ontwikkeling van ons project en die ons het meest hebben weerhouden om vooruitgang te boeken.

- Aan het begin van ons project waren we voor een periode van ongeveer een maand, niet in staat om sprites te tekenen op het scherm. We zochten namelijk naar een methode om de image om te zetten naar een array van kleurenindexen.

➔ Hierbij hebben we een poging gedaan om hiervoor een programma te schrijven in de programmeertaal 'C', maar achteraf gezien, na vele complicaties bleek dit niet de aangewezen taal te zijn (althans niet gemakkelijk om te realiseren in die taal). We deden nog een poging, maar dan in Python. Dit was een veel compacter en eenvoudiger alternatief, aangezien dit ons wel het gewenste resultaat gaf.

- We hadden ook een probleem in onze generische tekenprocedure zelf: we gingen over onze rijen in de sprite en daarbinnen gingen we over de kolommen waarbij we elke entry in de array afzonderlijk kopieerden naar het videogeheugen. (niet echt efficiënt)

➔ We hebben de binnenste lus weggehaald en vervangen door 'rep stosb' optimalisatie.

- Ook werd er in het begin telkens maar één rij van de sprite getekend op het scherm.

➔ We hebben de breedte van de sprite in pixels in elke iteratie laten initialiseren in een register, want als we dit niet deden, dan werd er dus maar één rij van de sprite getoond, aangezien dat register na elke rij van de sprite op nul werd gezet.

- Eerst hebben we geprobeerd om het raster van stenen te tekenen met één kleur. Voordat we ons spel lieten starten, wezen we een index toe aan elke steen zodat we achteraf op basis van deze index toch met meerdere kleuren konden werken.

➔ We merkten al snel op dat dit procedé eigenlijk nutteloos was, als we over onze stenen gaan om deze te tekenen, houden we eigenlijk al een index bij. Hierdoor hebben we een extra veld kunnen vermijden in onze datastructuur voor stenen.

- Voor het checken van het raken van de bal met de stenen zijn we een beetje te optimistisch geweest en dachten we dat dit niet zo heel moeilijk ging zijn maar uiteindelijk moesten we eigenlijk heel wat condities checken om het volledig werkende te krijgen.

➔ De enige oplossing hier was gewoon door zoveel mogelijk te testen en te zien waar het misliep, om dan vervolgens extra condities toe te voegen.

5 Conclusie

Over het algemeen hebben we feitelijk heel wat tijd verloren aan het tekenen van onze sprites. Het was dan ook moeilijk om al met andere dingen te beginnen voordat het tekenen van de sprites in orde was. Nadat dit was opgelost hadden we niet veel tijd meer over en hebben we ons volledig ingezet om de basisfunctionaliteiten op punt te krijgen en dit is ons dan uiteindelijk nog gelukt. We hebben niet echt een uitbreiding, maar met extra tijd zouden we zeker bereid zijn geweest om die te implementeren.