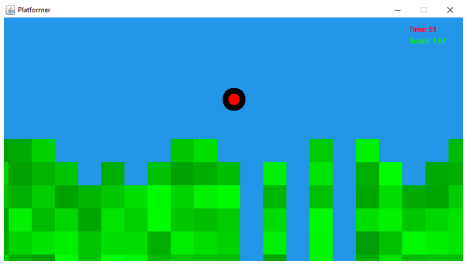
1 Praktijk 1

1.1 Inleiding

Nu wij in de vorige hoofdstukken de nodige achterliggende theorie hebben onderzocht, gaan we onze kennis inzetten voor het maken van twee praktijkopdrachten: een spel die een programma moet leren spelen en een programma om handgeschreven nummers te herkennen. In dit hoofdstuk zijn wij bezig met de vraag *Welke onderdelen zijn nodig voor een zelflerend computersysteem, in de vorm van een door ons ontworpen computerprogramma, dat in staat is een eenvoudig, zelfgemaakt computerspel, met daarin meerdere obstakels, te spelen?*

1.2 Het spel

Voordat we het zelflerende deel van het programma kunnen maken, moet er eerst een spel komen. Hiervoor kiezen wij een *platformer*. Er is een wereld waar de speler doorheen kan springen. De wereld wordt niet volgens een vast patroon gegenereerd. Er zitten heuvels, kuilen en gaten in. Wanneer de speler in zo'n gat valt, gaat hij dood. Het doel van het spel is simpelweg zo ver mogelijk te komen. Lopen doe je automatisch, dus het komt echt aan op de timing van je sprong. Voor een mens is dit makkelijk te bevatten, maar voor een computer kan dit een groot obstakel blijken te zijn. In figuur 2 is goed te zien hoe het spel is opgebouwd uit 'blokjes'. Op sommige blokjes, de groene, kan de speler staan. Alles wat blauw is, is niet solide en de speler valt er dus doorheen. (Dit is een eerdere versie van het spel)



Figuur 1: De eerste versie van het spel

1.3 Evolutionary improvement toepassen

Nu is het tijd om een zelflerend systeem te maken die dit spel kan leren spelen. We doen dit met evolutionary improvement (zie hoofstuk 5). Eigenlijk moet het programma een enkele vraag kunnen beantwoorden: wat is het beste moment om te springen?

Elk individu dat het spel speelt heeft DNA. In dit DNA staat beschreven hoe lang een speler telkens moet wachten voordat hij springt. (Ook staan de kleuren van het blokje en van de ogen hierin beschreven, maar dat heeft geen invloed op het spelen van het spel) Het DNA van de best presterende individuen

kan worden doorgegeven tussen generaties. Het idee is dat het programma zo steeds beter wordt doordat het steeds 'beter' DNA voor zijn spelers heeft. Ook kunnen er willekeurige mutaties en *crossovers* plaatsvinden. Een mutatie is een willekeurige verandering in het DNA en een crossover is het op twee verschillende plekken wisselen van de informatie op het DNA. Deze acties worden ondernomen om diversiteit in de populatie te vergroten. Stel je voor dat je op tijdstip 1,0 moet springen, maar geen enkel individu heeft hier de code voor, dan zal er nooit vordering kunnen komen.

De *fitness* van een bepaald individu wordt bepaald door zijn horizontale positie. Komt het individu verder, dan is hij meer geschikt en zal hij dus een hogere fitness hebben. Om tot het getal te komen dat weergegeven wordt als de fitness, wordt de positie echter in het kwadraat gedaan. Dit is niet zonder reden: op het moment dat een bepaald individu wél over een gat springt, maar een ander individu niet, dan is het verschil in x-positie niet zo groot, een gat is immers maar 1 of 2 blokjes breed. Om een kleine vooruitgang meer impact te geven op de fitness, kwadrateren we.

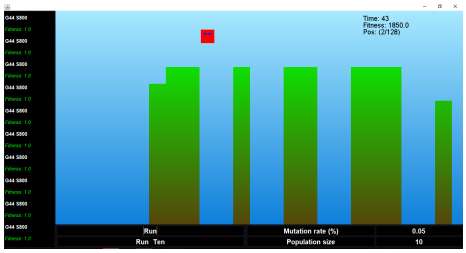
1.4 Het programma

Zodra je het programma opstart, kan je eerst een keer zelf het spel spelen. (springen doe je door op de 'w'-toets te drukken) Op het scherm is nu echter een stuk meer te zien dan alleen het spel. Rechtsonder zijn de *mutation rate* en de *population size* weergegeven, deze getallen geven respectievelijk aan hoe groot de kans is op een mutatie en de grootte van n generatie aan individuen.

Er zijn ook nog twee knoppen te zien: *Run* en *Run Ten*. Wanneer je op *Run* drukt, maakt het programma n generatie individuen aan en laat al deze individuen het spel doorlopen. De generatie staat links weergegeven, op volgorde van hoogste naar laagste score fitness. De naamgeving is als volgt: eerst een generatienummer (bijvoorbeeld G10 of G16) en daarna het nummer van het individu uit de generatie (S102, S300, enzovoorts). Als je op een bepaald individu klikt, wordt er afgespeeld hoe hij het spel speelt.

Wanneer je nogmaals op *Run* drukt, wordt de volgende generatie aangemaakt. De individuen uit de nieuwe generatie kunnen dus wat DNA 'overerven' van de vorige. Links wordt de nieuwe populatie weergegeven. We verwachten hier een hogere fitness te zien, doordat het programma steeds zal proberen het beste DNA over te geven.

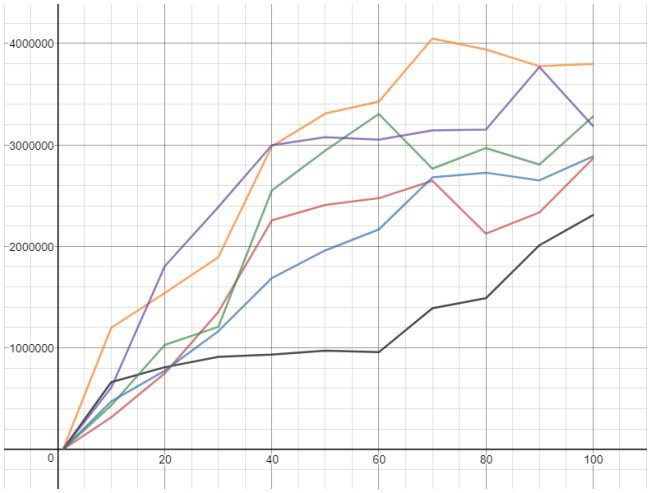
Wanneer je op *Run Ten* drukt, loop te simulatie tien maal en je krijgt de resultaten te zien van de laatste generatie.



Figuur 2: Een bepaald individu dat, aangestuurd door het programma, het spel speelt

1.5 De resultaten

Het is de bedoeling dat het programma beter wordt in het spelen van het spel, naarmate er meer generaties zijn geweest. En ja hoor! Wanneer je het n run doet, zie je al een grote verandering in de fitness. Druk je vervolgens nog een aantal keer op *Run Ten*, dan zie je de getallen nog groter worden. In figuur 3 is de fitness te zien van het beste individu van 10 generaties uit 6 verschillende tests (Langs de horizontale as staat het generatienummer, langs de verticale staat de fitness). Ook hier is goed te zien dat het programma alsmaar beter wordt. Soms lijkt er generaties lang echter geen vordering te zijn, soms neemt de fitness zelf af, maar vaak komt het programma ook zo'n dipje uit. Dit kan als volgt verklaart worden: het is waarschijnlijk dat er op een bepaalde plek in de wereld, waar de individuen doorheen moeten, een moeilijk obstakel zit, waar nog geen enkel individu het DNA voor heeft om overheen te komen. Nu moet het programma geluk hebben dat er een mutatie plaatsvind, die helpt dit obstakel te overkomen. Bij de rode lijn is bijvoorbeeld te zien dat er rond generatie 80 een daling in fitness plaatsvond, maar ook dat er daarna ook weer vordering kwam.



Figuur 3: De hoogste fitness per 10 generaties voor 6 verschillende tests. (Grafiek gemaakt m.b.v Desmos LINK DESMOS HIER)

1.6 Conclusie

Een computerprogramma blijkt niet erg intelligent te hoeven zijn om zichzelf een spel als deze te kunnen leren spelen. Door middel van willekeurige trial and error kan het programma steeds beter worden. Wij hebben gekozen dit evolutionair te doen, maar we denken dat dit ook op de andere in hoofdstuk 5 genoemde manieren kan. We hebben gemerkt dat de speler, aangestuurd door het programma, zich niet 'bewust' hoeft te zijn om de wereld waarin hij loopt, hij hoeft immers alleen te weten wanneer hij over een gat heen moet springen.