Universiteit van Amsterdam

Jim Wagemans – Steven de Rooij

Opdracht 5: The Maze Runner

1 Introductie

Bij deze opdracht ga je zelf een spel maken. De opdracht bestaat uit twee delen.

In het eerste deel ontwikkel je de functies die zijn voorgeschreven, wat uitmondt in een eerste eenvoudig doolhofspelletje.

In het tweede deel ben je volkomen vrij om het spel verder uit te breiden zoals je zelf leuk vindt. We geven wel een aantal tips en suggesties voor toevoegingen die je relatief gemakkelijk zou kunnen maken. Aan het einde van de cursus maken we de spelletjes van je medestudenten beschikbaar op Canvas :)

1.1 Deadlines en beoordeling

Deze opdracht bestaat uit twee delen, die elk een aparte deadline hebben en die allebei afzonderlijk tellen als 20% van je practicumcijfer.

Het eerste deel is vergeleken met eerdere opdrachten niet bijzonder groot, dus we raden sterk aan om als het lukt al voor de deadline aan deel 2 te beginnen.

De beoordeling voor deel 2 werkt opnieuw hetzelfde als je gewend bent, behalve dat omdat dit deel zo vrij is er deze keer geen autotests zijn. De 5 punten voor autotests zijn in plaats daarvan over twee categorieën verdeeld:

- Functionaliteit: Is er veel toegevoegd? Zijn er moeilijke dingen toegevoegd? Werkt het spel zoals bedoeld? Zijn er veel ruwe kantjes?
- Documentatie en spelontwerp: is er documentatie die uitlegt hoe je het spel moet spelen? Is de presentatie uitnodigend en laagdrempelig?

Voor elke categorie verdien je 0 punten als je programma in die categorie onder de maat is; 1 punt voor een voldoende poging; 2 punten voor een goede poging, en 3 punten als je spel in die categorie excelleert.

Belangrijk: als je minder dan 5 punten krijgt op deze twee categorieën haal je dus minder dan een tien voor deel 2. Dat betekent dan *niet* per se dat we vinden dat er iets op je spel aan te merken is. Bij creatief werk is beoordeling altijd enigszins subjectief, dat kunnen we niet vermijden als we ruimte willen overlaten om excellentie te belonen. Als je een uitzonderlijk goed spel inlevert is het mogelijk om tot 11 punten te halen voor deel 2, maar dit is onwaarschijnlijk. Ook als je voor eerdere opdrachten tienen hebt gehaald kan het gebeuren dat je voor dit onderdeel een lager cijfer haalt; dat is dan niet noodzakelijkerwijs kritiek.

2 Deel 1: een eenvoudig doolhof

2.1 Een beginnetje met `ncurses`

Om een interactief spel te kunnen maken gebruiken we een heel eenvoudige library: `ncurses` (`https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/`). Daarmee kun je ASCII symbolen op het scherm afdrukken, eigenlijk zoals je ook met printf al kon doen, maar nu kun je ze ook op zelfgekozen posities laten verschijnen, en je kunt ook direct reageren op toetsindrukken, in plaats van dat je daar pas iets van merkt als de speler op Enter drukt.

Het zou goed kunnen zijn dat de library `ncurses` al op je systeem is geïnstalleerd. Je kunt dat controleren door al eens `make voorbeeld` aan te roepen. Als dat gewoon werkt, dan is `ncurses` beschikbaar. Zo niet, dan doen we een beroep op je zelfredzaamheid. Installeren is niet moeilijk maar hangt natuurlijk wel een beetje af van de details van je systeem. Je moet dan de volgende stappen volgen:

- Op een Mac: installeer (als je die niet al hebt) eerst Homebrew (`https://brew.sh/`). Je kunt daarna met `brew install ncurses` de library installeren.
- Op Linux: installeer de library met `sudo apt install libncurses5-dev`.
- Als er iets niet werkt: google, of vraag hulp bij een medestudent of op het practicum.

Als `ncurses` geïnstalleerd is kun je `make voorbeeld` uitvoeren. Kijk maar eens wat het programma doet, en kijk of je de code van `voorbeeld.c` kunt lezen. Zoek eventueel meer informatie via de link hierboven of met google.

2.2 Het voorbeelddoolhof

Je spel zal een doolhof inlezen uit een tekstbestand. Er staat een voorbeeld van zo'n doolhof in de subdirectory `assets` dat er zo uitziet:

```
######################
#*#                  #
# #  $   ###########  #
# ########           #
#        # ############
# ###### #           #
# #    # ############ #
# #### #     #        #
#    # ##### #  XXXXXX#
# XXX#       #        #
#    #########XXXXXXX #
#  X                  #
######################
```

Het idee is dat de speler begint op de positie van de asterisk (`*`). Het doel is te bewegen naar de uitgang (`$`). Het probleem is dat de speler niet kan bewegen door muren (`#`), en zelfs verongelukt als zij in contact komt met een val (`X`).

Een groot deel van de opdracht bestaat er uit om te bedenken hoe je zo'n doolhofbestand kunt inlezen, en de functies te maken die de speler in staat stellen door het doolhof te bewegen en dergelijke. Dit wordt allemaal gedaan door de module `rooster`.

2.3 Het uitwerken van de `rooster` module

De toestand van het spel wordt opgeslagen in een nieuw datatype `rooster`. Alle dingen die in het spel kunnen gebeuren veranderen het `rooster` op de een of andere manier.

Alle functies die een `rooster` moet ondersteunen zijn gedeclareerd in het interface: `rooster.h`. **Verander dit bestand niet!** De definitie van het datatype, en alle relevante functiedefinities, moeten grotendeels door jou worden ontworpen in `rooster.c`.

Je moet zelf nadenken over welke velden je in `struct rooster_data` moet stoppen om te zorgen dat je alle functies makkelijk kunt ondersteunen. Spelobjecten zullen we voor het gemak in het rooster opslaan als `char`, net zoals in het voorbeelddoolhof. Je hebt verder vrije keuze hoe je het doolhof representeert. Zorg dat je in `struct rooster_data` op een makkelijke manier hebt opgeslagen wat de afmetingen zijn van het speelveld, wat de huidige toestand is van het spel, en welk object er op bepaalde coördinaten staat.

- **Opdracht (1pt).** Maak een bestand `rooster.c` en schrijf daarin een definitie van het datatype `struct rooster_data`. Vergeet ook niet om de header in te lezen met `#include`.

Als je definitie van de struct hebt voltooid kun je de functies implementeren. Het is handig om alle declaraties uit de header te kopiëren naar `rooster.c`, dan weet je precies welke functies je nog moet implementeren en heb je alle commentaar makkelijk bij de hand.

Als je het handig hebt aangepakt is eigenlijk alleen de functie `rooster_lees` lastig, tenminste, als je het echt goed wilt doen. Het lastige zit hem er namelijk in om goed om te gaan met mogelijke foutcondities. Bijvoorbeeld, al het geheugen dat je al hebt opgevraagd met `malloc` moet je weer vrijgeven als er een fout optreedt. Neem dus de tijd om deze functie zorgvuldig uit te denken.

Hint: de makkelijkste manier om het doolhof in te lezen is door te onderzoeken hoe groot het doolhofbestand is, dan met `malloc` ruimte te alloceren voor een char array, en het hele bestand in een keer in te lezen. Je moet daarna nog wel onderzoeken of alle rijen even lang zijn en hoe veel rijen en kolommen er zijn, en goed met fouten omgaan. Als je dit op deze manier doet, dan kun je uitrekenen op welke positie in het array een object op een bepaalde positie staat. Houd er daarbij rekening mee dat er na elke rij nog een newline karakter `'\n'` staat dat niet bij het doolhof hoort. Als je doolhof array bijvoorbeeld 20 rijen van 30 objecten heeft, dan staat het object op positie $(7, 3)$ opgeslagen op index $7 + 3 * (30 + 1) = 100$ van het array. Een alternatief is om het doolhof regel voor regel in te lezen met `fgets`.

- **Opdracht (2pt).** Implementeer alle functies die gedeclareerd worden in `rooster.h` in `rooster.c`.

2.4 Spellogica toevoegen

Nu we de fundering hebben gelegd kunnen we aan het echte spel beginnen. In het bestand `spel.c` vind je een beginnetje van de spellogica.

- **Opdracht (1pt).** Kijk af bij `voorbeeld.c` hoe je `ncurses` kunt gebruiken (en/of zoek online naar extra uitleg), en maak de functie `toon_rooster` in `spel.c`. Je hoeft hierbij eigenlijk alleen de `ncurses` functie `addch` te gebruiken. Kijk vervolgens of het programma is aangeroepen met als argument de naam van een doolhofbestand. Zo ja, lees dit doolhof dan in en test de functie `toon_rooster`. Gebruik het voorbeelddoolhof om te proberen of het werkt.

Deze versie van het spel is heel eenvoudig: aan het begin van het spel wordt de toestand van het rooster op `AAN_HET_SPELEN` gezet. Elk doolhof moet precies één speler object `'*'` bevatten. De speler kan met de pijltjestoetsen (en/of W, A, S, D als je dat liever wilt) door het doolhof lopen: dat wil zeggen, door de gangen (object `' '`). Door muren (object `'#'`) wordt de speler geblokkeerd. Als de speler op een valse tegel (object `'X'`) stapt komt hij ongelukkig aan zijn einde (de toestand van het rooster wordt dan op `VERLOREN` gezet). Als de speler de schat (object `'$'`) vindt is het spel gewonnen (de toestand wordt dan op `GEWONNEN` gezet).

Het spel gaat door zolang de toestand van het rooster `AAN_HET_SPELEN` is. Als het spel is afgelopen wordt een boodschap afgedrukt dat het spel is gewonnen of verloren.

Je hebt in het voorbeeld al gezien dat je met de functie `getch` kunt wachten op een toetsaanslag, en dat in `ncurses.h` macro's zoals `KEY_LEFT` worden gedefinieerd met integer waardes voor de verschillende toetsen.

- **Opdracht (1).** Roep op basis van welke toets is ingedrukt de functie `beweeg` aan met de juiste waardes voor `dx` en `dy`. (De declaratie en documentatie voor deze functie is te vinden in `spel.c`.)
- **Opdracht (1).** Implementeer de functie `beweeg` zodat het spel wordt uitgevoerd precies zoals hierboven beschreven.

Als je dit af hebt is deel 1 van de opdracht voltooid! Maak een tar bestand met `make tarball1`. Lever je werk in en ga meteen verder met deel 2.

3 Deel 2: je eigen spel

Kopieer voor je aan dit deel begint je directory deel1 als deel2. (Dit kun je doen met `cp -r deel1 deel2`.) Werk vervolgens in de directory `deel2`.

3.1 Wat mag je veranderen?

Het antwoord op deze vraag is eenvoudig: *alles*, maar sommige dingen moeten in overleg met een TA. Je mag in ieder geval elke file behalve de `Makefile` aanpassen. Je mag er zelfs een totaal ander soort spel van maken. De enige harde eis is dat het in correcte C geprogrammeerd is, volgens de richtlijnen in deze cursus. Gebruik in principe ook geen libraries gebruikt behalve `ncurses` en de C standaard library. Uitzondering: in overleg met een TA kunnen ambitieuze studenten ervoor kiezen om de game library `sdl` te gebruiken om een nog mooier spel te kunnen presenteren. In dat geval mag je de `Makefile` aanpassen om de `sdl` library mee te linken.

Het is wel belangrijk om te bedenken dat veel veranderingen moeilijker zijn dan je misschien eerst zou denken. Nieuwe dingen bedenken duurt lang. Je hebt ook niet onbeperkt veel tijd voor deze opdracht. Om die reden hebben we wel een aantal adviezen:

- Dit soort opdrachten kunnen uitnodigen tot extreme prestaties. Geef daar niet teveel aan toe, hou jezelf in de hand! Je bent echt niet verplicht om iets gigantisch te maken om een redelijk cijfer te halen. Kies iets wat je zelf leuk vindt en wat naar jouw eigen inschatting voor jou echt goed te doen is in de tijd die voor de opdracht staat. Maak je er niet te veel zorgen over dat sommige andere studenten misschien meer kunnen, daar houden we rekening mee in de beoordeling.
- We raden sterk aan het principe te behouden van een spel gemaakt met `ncurses`, waarin je door toetsindrukken de toestand van een rooster kunt beïnvloeden, en waarin er niets gebeurt zolang je geen toets indrukt.
- Denk in eerste instantie aan uitbreidingen die bestaan uit het toevoegen van nieuwe soorten objecten, en denk op voorhand na over of je ook echt voor je ziet hoe je de uitbreiding die je hebt bedacht zou kunnen implementeren.
- Probeer ook een meer inhoudelijke uitbreiding in het spel aan te brengen. Hieronder vind je een aantal suggesties. Je moet proberen iets te maken waar je trots op kunt zijn.
- Schrijf voor jezelf op welke uitbreidingen je wilt doen, en schrap alles wat niet strikt noodzakelijk is. Implementeer wat overblijft eerst; als je dan nog tijd over hebt kun je een tweede uitbreiding bedenken. Dat is beter dan teveel hooi op je vork nemen en uiteindelijk niks af krijgen.
- Voel je niet te goed om de concrete suggesties voor uitbreidingen van hieronder over te nemen. Als je dit doet krijg je misschien geen “excellent” voor spelontwerp, maar “goed” kun je er zeker mee halen. Het is eigenlijk de veiligste strategie voor een redelijk cijfer.

3.2 Programmastructuur

Als je niet oppast kan je programma bij het toevoegen van nieuwe functionaliteit vrij makkelijk rommelig worden. Probeer verschillende functionaliteiten van elkaar gescheiden te houden en niet alles tegelijk te doen. Stop zo mogelijk elke nieuwe wijziging in zijn eigen functie.

3.3 Extra .c en .h bestanden

Deze mag je gewoon toevoegen in dezelfde directory als `spel.c`. Ze worden dan vanzelf meegenomen door de `Makefile`. De `Makefile` zelf mag je niet veranderen.

3.4 Assets

Als je extra bestanden met je spel wilt meeleveren (bijvoorbeeld data voor verschillende levels), dan kun je die opslaan in de subdirectory `assets`. Alle bestanden die je in die directory opslaat zullen vanzelf worden opgenomen in de tarball die je inlevert. **Belangrijk:** als je databestanden gebruikt die niet in

de `assets` directory staan komen die niet vanzelf in je tarball te staan en vergeet je dus misschien om ze in te leveren.

Omdat we willen dat je spel ook door ons en je medestudenten te spelen is, willen we je vragen om in het bestand `assets/handleiding.txt` een korte uitleg te geven van hoe je spel werkt. (Daarom is dit ook een onderdeel van onze beoordeling voor spelontwerp.) Je kunt ook kiezen om die uitleg in het spel zelf op te nemen; schrijf dan in `handleiding.txt` even kort dat de uitleg voor het spel in het spel zelf te vinden is.

3.5 Inleveren

Het is makkelijk om een onderdeel van je spel te vergeten in te leveren, maar dat kan wel punten kosten. Let daarom goed op het volgende:

- Gebruik `make tarball2`, niet per ongeluk `make tarball1`.
- Zorg dat alle extra bestanden die je gebruikt echt in `assets` staan.
- *controleer* je tarball! Om dit te doen kopieer je de tarball naar een lege directory, en dan typ je in `tar xzf tarball2.tar.gz`. Nu kun je zien of je spel compileert en alles bevat wat hij moet bevatten. Doe deze controle echt! We kunnen niet achter iedereen aanrennen om ontbrekende bestanden toch nog te krijgen.

3.6 Suggesties

Hieronder een aantal suggesties van mogelijke eenvoudige toevoegingen.

- Een level kan meerdere `'$'` objecten bevatten, en je wint als je ze allemaal hebt verzameld.
- Maak objecten `'|'` en `'-'` die twee richtingen blokkeren als een muur, maar vanaf de andere twee richtingen kun je op het object stappen en het zo vernietigen.
- Maak een valhek `'"'`. Als je er onderdoor loopt valt het en verandert het in een muur.
- Maak een object `'O'`: een blok dat je kunt verschuiven mits de locatie *achter* het blok vrij is. (Je kunt dit als basis gebruiken om het bekende puzzelspel Sokoban te implementeren: zie `https://en.wikipedia.org/wiki/Sokoban`).
- Maak "the blob": een object dat zich af en toe willekeurig uitbreidt naar lege naburige posities in het rooster.
- Maak dat het spel uit meerdere levels bestaat.
- Maak dat je de inhoud van het doolhof alleen kunt zien in een beperkte straal rondom de speler. (De speler heeft een fakkel.)
- Maak dat de speler altijd in het midden staat en dat als je beweegt het doolhof verschuift in plaats van de speler.
- Voeg twee objecten toe waarmee je spel Conway's Game of Life kan simuleren.
- Geef het spel een beginscherm, met de handleiding van het spel en eventueel een animatie in ASCII.

Lastiger, maar wel de moeite waard:

- Maak een of meerdere monsters die willekeurig rondlopen, of volgens een vast patroon. (Het monster zet een stap bij elke toetsaanslag.)
- Maak trappen waarmee je naar andere verdiepingen kunt gaan.
- Maak een "editor", waarmee je zelf interactief nieuwe levels kunt ontwerpen en opslaan.
- Ontwerp een strategisch spel en maak een computergestuurde tegenstander. (Bijvoorbeeld dammen of Othello/Reversi.)
- Maak het spel (ondanks onze waarschuwingen) real-time.

- Voeg code toe om (stukken van) het doolhof automatisch te genereren.

Er zijn natuurlijk nog veel meer mogelijkheden, en het leukste is als je iets bedenkt waar we nog niet aan gedacht hadden. Veel succes!