Structuur van computerprogramma's II

Documentatie: Tetris (augustus 2016)

Asma Oualmakran

Inhoudsopgave

Introductie

In het volgend rapport bespreken we een eigen implementatie van het klassieke spel Tetris. In dit spel is het de bedoeling om dalende blokjes in het speelveld op een manier te plaatsen waardoor de hele breedte wordt ingenomen door deze blokjes. Wanneer dit het geval is zal de gevulde rij verdwijnen, dalen de rijen erboven en verdient de speler punten. Wanneer de speler de bovenkant raakt en het blokje niet meer naar beneden kan bewegen, dan is het spel afgelopen.
In deze implementatie is er gebuik gemaakt van de taal C en de SDL libraries. Die libraries worden in het project gebruikt om de GUI te voorzien en om user input mogelijk te maken.
In deze implementatie is het mogeklijk om blokjes af te beelden in de GUI en via deze user input te manipuleren. De volgende manipulaties zijn mogelijk, rotatie, naar links bewegen, naar rechts bewegen, naar beneden bewegen.
Er is ook een hold voorzien waarbij het volgend blokje wordt opgeslagen en gespawnd nadat het huidig blokje niet verder naar beneden kan bewegen, echter wordt de hold niet correct weergegeven in de GUI.
De blokjes worden bij initiatie random gekozen, zowel bij initiatie van het eerste blokje als bij het inladen van een nieuw blokje in de hold.
Bovendien zijn het aantal blokjes uitgebreid, het spel bevat niet enkel de klassieke vormen, maar ook nog drie “nieuwe vormen”.
Het spel bevat ook de functionaliteit om de score te verhogen, de hoogste score op te slaan, een volle lijn verwijderen, deze zijn voorzien en werken, maar worden niet gebruikt in het spel.
Het spel kan nog niet worden opgeslagen of worden gepauzeerd.

Architectuur

In onderstaande figuur worden de afhankelijkhede voorgesteld. Hierbij zijn de objecten waar de pijl aankomt afhankeljik van de objecten waaruit de pijl vertrekt.

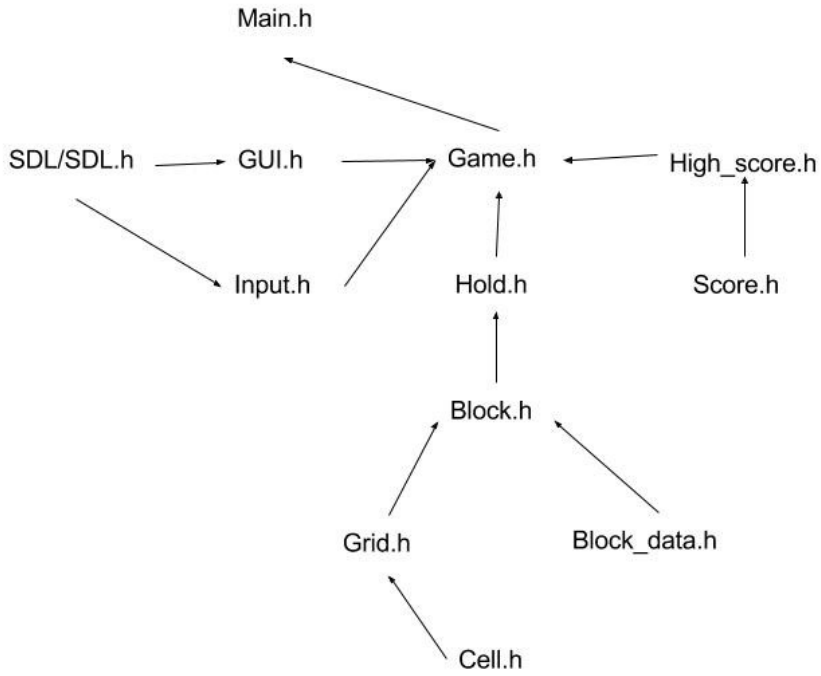

Illustration 1: Afhankelijkheden

In het ontwerp is ervoor gekozen om de *GUI* (graphical user interface) en de *user input* volledig van elkaar gescheiden te houden. Dit zorgt er namelijk voor dat men makkelijk een andere GUI of andere manier van user input ontvangen kunnen voorzien, wat de code ook makkelijker aan te passen is.

In de gekoze architectuur is ervoor gekozen om de delen op te bouwen vanuit basis elementen.

Code

De code is opgesplitst in verschillende .C en .H files waarbij in de .C de definities van de functies bevatten, en de .H files de declaraties van de functies, data-structuren en constanten die worden gebruikt.

Main

Void main(void);

Deze file bevat de main-functie, waar na de compilatie van de code de main funtie wordt opgeroepen.

Block

Dit ADT wordt gebruikt om de tetris blokjes voor te stellen, initialiseren en om ze te kunnen manipuleren in het spel. Hierbij wordt zo'n blokje voorgesteld door een struct, die alle gegevens bijhoudt van dat blokje. Een struct bestaat uit de volgende gegeves:

– `struct Cell cell` → een cell struct die in Cell verder zal worden besproken

Om de locatie te bepalen van een block wordt er gebruik gemaakt van een pivot, deze pivot heeft een bepaalde coordinaat in het speelveld en een in de rotatie matrix.

– `int x_piv` → de x-coördinaat van de pivot in het speelveld
– `int y_piv` → de y-coördinaat van de pivot in het speelveld

deze twee integers vormen samen een locatie in het speelveld.

– `int rotation_matrix[MATRIX_WIDTH][MATRIX_HEIGHT]` → de rotation_matrix houdt de vorm en de rotatie van het blockje bij, deze wordt uit Block_data.h gekopieerd en de voorstelling zal worden besproken in Block_data.
– `enum Colour colour` → hier wordt de kleur bijgehouden aan de hand van een enumeration die ook in Block.h staat gedeclareerd

– `enum Shape shape` → hier wordt de vorm van het blockje bijgehouden aan de hand van een enumeration die ook in Block.h wordt gedeclareerd.

Aangezien enkel de locatie van de pivot in het speelveld wordt bijgehouden, gaat men voor het bepalen van de locatie van de andere blokjes binnen de figuur, de afstand tussen het bepaalde blokje en de pivot berekenen in de rotatie-matrix (door eerst de locatie van de pivot binnen dezelfde matrix te bepalen) en die afstand wordt dan gebruikt de locatie binnen het speelveld te bepalen.

Block_data

In deze file wordt er een array bijgehouden die bestaat uit multi-dimensionale arrays. In deze multi-dimensionale arrays zitten de voorstellingen van de blokjes, hierbij wordt er gebruik gemaakt van de getallen 0,1,2. Hierbij stelt 0 een lege locatie voor, 1 een gevulde locatie en 2 de locatie van de pivot in de multi-dimensionale array en dus ook de pivot van de figuur.

Cell

De `struct Cell` is de basis component van het speelveld grid, iedere locatie van het speelveld wordt voorgesteld door deze struct.
Die uit volgende zaken bestaat:

– `enum State state` → dit stelt de status voor van de cel, het kan gevuld zijn, leeg of een pivot bevatten

– `enum Colour colour` → dit stelt de kleur voor van de cell

– `int x_pos` → dit houdt de x-positie van de cell bij

– `int y_pos` → dit houdt de y-positie van de cell bij

GUI

Dit ADT is geimplementeed om de grafische interface te voorzien, hierbij wordt er een venster aangemaakt en de oppervlakte van dat venster aangepast zodat we de ontwikkeling in het speelveld te zien krijgen.
Hierbij wordt er gebruik gemaakt van de SDL libraries. Wat de speler nu tezien krijgt is het speelveld en het blokje dat de speler krijgt. De hold wordt ook afgebeeld in de GUI maar deze geeft geen correcte weergave van welk blokje in de hold aanwezig is.

Game

In dit ADT komen alle elementen samen, hierin worden de testen uitgevoerd die bepalen of een blokje de bodem raakt, of het mogelijk is om het blokje te doen bewegen. Dit ADT zorgt er ook voor dat het speelveld steeds wordt aangepast bij iedere iteratie en dus ook de correcte weergave geeft van wat er aanwezig is in het speelveld.
De clear_line functie wordt hier ook opgeroepen, maar is niet getest wegens tijdsgebrek.
Game bestaat bovendien ook uit een struct die alle speel elementen bijhoud, zoals het blokje, de hold, het speelveld, het score bord, ect.
In deze implementatie worden alle elementen een maal aangemaakt, zoals de struct van het blokje. Deze wordt bij initiatie van het spel aangemaakt, en wanneer het blokje niet meer verder kan bewegen, worden de gegevens van de struct die de voorstelling van het blokje voorzien aangepast. Dit gebeurt door de gegevens van de struct die de hold voorsteld in de struct van het blokje wordt gekopieerd.

Grid

Het volgend ADT stelt het speelveld voor, het speelveld wordt gemaakt door twee multi-dimentionale arrays dynamish te alloceren en deze worden dan opgevuld met pointers naar struct Cell. Dit zorgt ervoor dat er op iedere locatie van het speelveld een struct Cell aanwezig is die dan kan worden aangepast. Hierbij worden de cellen gevuld of leeggemaakt, en krijgen ze wanneer ze gevuld zijn de kleur van het blokje wanneer deze leeg is wordt de cell zwart gekleurd.
Deze gegevens worden gebruikt om te bepalen of een block naar een bepaalde locatie kan bewegen, en om te GUI te kunnen tekenen.
Dezelfde gegevens kunnen worden gebruikt om het speelveld op te slaan, hierbij kan men de kleuren van de cellen gebuiken. In dit ontwerp zou men over het speelveld heen gaan en op iedere locatie de aanwezige kleur bepalen. De aanwezige kleuren worden dan weggeschreven in een bestand, die daarna ingelezen kan worden om de kleuren op de juiste locatie te plaatsen bij het laden van een opgeslagen spel.

High_score

Aan de hand van dit ADT gaat men de hoogste scores van gespeelde spellen bepalen en opslaan in een bestand. Bij het initialiseren van de high_score, wordt er een file uitlelezen waarin de scores aanwezig zijn en worden ze ingeladen in een array in de volgorde waarin ze zijn tegengekomen, in de file zijn de scores al gesorteerd van groot naar klein. Wanneer deze file leeg is, zal de array worden opgevuld met nullen aangezien er nog geen scores aanwezig zijn. Nadat ze zijn ingeladen, is de high_score geinitialiseerd en kan dus gebruikt worden. Nadat men het spel afsluit gaat men na of de behaalde score in de de array kan, dit betekend dat de score groter moet zijn dan de laagst behaalde score.
Zo niet wordt de score niet in de array gestopt, zowel wordt de score in de array gestopen en de array wordt gesorteerd. Hierna wordt de array weggeschreven naar de file die deze scores bevat. Wegens tijdsgebrek wordt dit ADT niet gebruikt en is er ook geen voorstelling van in de GUI.

Hold

Dit ADT maakt het mogeljk om het volgende blokje bij te houden en ook in te laden als blokje waarmee de speler kan spelen. Deze bestaat uit een struct die een struct Block bevat en een matrix waarin het volgend blokje wordt bijgehouden. Het volgend blokje wordt bepaald aan de hand van de standaard functie rand()%<int>, dit is echter niet volledig random en de de blokjes worden bij ieder nieuw spel in dezelfde volgorde geladen.
Zoals eerder vermeld zijn er problemen met het voorstellen van de hold in de GUI.

Input

Dit ADT is geimplementeerd aan de hand van de SDL library en maakt gebruik van events om user input te aanvaarden en te verwerken. Deze zal bij input de bijbehordende fucties oproepen om de blokjes te kunnen manipuleren, ook kan men het venster sluiten en is er een knop voorzien om het spel te pauzeren. Aangezien die functionaliteit nog niet wordt voorzien in het spel, wordt er niets gedaan bij het induwen van die toets.

Score

Dit ADT wordt gebuikt om de score bij te houden binnen het spel. Hierbij wordt deze iedere keer opgehoogt bij het vol krijgen van een rij.
Hoewel deze functionaliteit is voorzien wordt deze echter niet gebruikt in deze implementatie wegens tijdsgebrek

Testen

Het testen van de logica gebeurde via prints binnen de console, om output van de functies te kunnen lezen. Het debuggen van de GUI werdt gedaan door de combinatie van prints naar de console en de GUI te starten.

Conclusie

Er zijn nog steeds elementen die nog niet werken binnen deze implementaite zoals het gebruiken van de high-scores, scores, het verwijderen en doen dalen van rijen, het opslaan van een spel en het pauzeren van het spel. Deze zijn niet aanwezig wegens tijdsgebrek.

Mogelijke implementaties van de ontbrekende componenten:

– High-scores: De functionaliteit is aanwezig en werkt, het moet enkel nog worden geintegreerd met de grafische interface voor de weergave en integratie met het Game ADT voor manipulaties.

– scores: De functionaliteit is aanwezig en moet enkel nog worden geimplementeerd dat het wordt voorgesteld in de GUI en het wordt gebruikt door het ADT game.

– Het verwijderen van rijen: In het ontwerp zou men nadat een blokje niet meer verder naar beneden kan bewegen op de volledige hoogte van het blokje controleren of de volledige rij gevuld is. Hierbij test men van beneden naar boven, en wanneer men een lege cell tegenkomt gaat men de test opnieuw beginnen voor de rij erboven.
Er wordt ook een constante bijgehouden die dan zal tellen hoeveel rijen er al vol zijn, en deze kan worden gebruikt om de score op te hogen. En weet men ook hoe ver men de bovenstaande niet volle rijen kan doen dalen.
Het doen dalen van de rijen, zal gebeuren door de toestanden en de kleuren van de cellen te overschrijven met de gegevens van de bovenstaande cellen.