Universiteit Gent

Systeemprogrammeren

Projectopgave

9 oktober 2017

Inhoudsopgave

1 Inleiding

Alvorens de volledige opgave van het project toe te lichten, wordt eerst een korte inleiding gegeven over een aantal deelaspecten van het project. Het is nuttig om dit door te nemen en zo een goede achtergrondkennis te hebben voordat er aan de eigenlijke opgave begonnen wordt.

1.1 Algemene projectinformatie

Dit project loopt gedurende het volledige eerste semester en moet in een groep van vier personen uitgevoerd worden. Het project heeft als focus zoveel mogelijk aspecten van software development en systeemprogrammeren aan het licht te laten komen. Hierbij denken we behalve het programmeren zelf, aan volgende aspecten: collaboration & tooling, testing, integratie van bibliotheken, documentatie & API's doornemen, design en implementatie van algoritmes, etc.

Gedurende de ontwikkeling heeft elke groep een assistent die jullie zal begeleiden. Bij vragen of problemen kan die jullie verder helpen. Ook zal op geregelde tijdstippen een meeting ingepland worden met de volledige groep. Bedoeling van deze meetings is dat jullie kort een overzicht geven van de status van het werk (bvb. door middel van een korte presentatie). Vragen die jullie tijdens de meeting opgelost wensen te zien worden best vooraf naar de assistent gestuurd zodat die ze kan doornemen vóór de meeting.

Het einddoel van dit project is een werkende, platform-onafhankelijke C/C++ implementatie van een spel af te leveren. Er is echter ook een deadline halverwege, waarbij het afgewerkte C gedeelte ingediend moet worden. We noemen het spel "Pi Escape 2", het is zeer los gebaseerd op bestaande puzzle games. Het spel is eenvoudig gehouden, en is dus geen afgewerkt spel. Aan welke voorwaarden het spel precies moet voldoen, zal in de volgende secties volledig uit de doeken gedaan worden.

Lees ook zeker de sectie over opzetten van GitHub en indienen (Secties 7.1 & 7.5), want jullie vooruitgang wordt gecontroleerd via een GitHub-repository van UGent. We gaan er van uit dat alle collaboratie en documentatie online gebeurt. Beperk dus het gebruik van online platformen (e.g. Slack,

Discord) waar wij geen toegang tot hebben. Als je alsnog wenst gebruik te maken van dergelijke platformen, zorg dan dat de informatie / beslissingen ook via GitHub beschikbaar gemaakt worden.

Besteed voldoende tijd en aandacht aan dit project, want het gaat om 5 van de 20 punten voor het vak Systeemprogrammeren die je kan winnen, maar dus ook verliezen.

Dit project is een groepswerk, en het is de bedoeling dat iedereen evenveel bijdraagt. Tijdens de meetings en via GitHub zal hier over gewaakt worden. Zoniet zal dit in rekening gebracht worden bij de toegekende score.

1.2 Game programming

Spellen programmeren is op een aantal vlakken anders dan een gewone applicatie programmeren. In deze sectie worden een aantal basisaspecten van game development uit de doeken gedaan.

1.2.1 De game loop

De centrale component van eender welk spel is de game loop. De game loop zorgt ervoor dat het spel vlot blijft lopen, onafhankelijk van het feit of de gebruiker al dan niet input genereert. Meer traditionele softwareprogramma's reageren op gebruikersinput en doen niets zonder deze input. Neem bijvoorbeeld een tekstverwerker. Deze formatteert woorden en tekst terwijl de gebruiker typt. Als de gebruiker niets typt, dan doet de tekstverwerker niets. Sommige functies mogen dan wel lang duren om uit te voeren, ze zijn allemaal gestart door een gebruiker die het programma vroeg om iets te doen.

Spellen moeten daarentegen constant blijven uitvoeren, onafhankelijk van de al dan niet aanwezige input van een gebruiker. Een game loop in pseudocode zou er zo kunnen uitzien:

```
while (user doesn't exit)
     check for user input
     move & run AI
     resolve collisions
     draw graphics & play sounds
```

```
end while
```

Hierbij wordt zolang de gebruiker het programma niet stopt, achtereenvolgens gecontroleerd welke input er gegeven is, de beweging in het spel uitgevoerd, de artificiële intelligentie (AI) van de vijand en logica van het spel uitgevoerd, worden eventuele botsingen (*collisions*) opgelost, de beelden getekend en de geluiden afgespeeld. De game loop kan verder verfijnd en aangepast worden terwijl de ontwikkeling van het spel voort gaat, maar de meeste spellen zijn op dit basisprincipe gebaseerd.

Game loops kunnen verder nog verschillen afhankelijk van het platform waarop ze ontwikkeld zijn. Als voorbeeld, een spelletje voor DOS en consoles kon alle processing resources voor zijn eigen rekening nemen en gebruiken naar wens. Een spel voor een Microsoft Windows besturingssysteem, moet uitgevoerd worden binnen de beperkingen van de process scheduler.

Verder maken de meeste huidige spellen gebruik van multi-threading, zodat bijvoorbeeld de berekening van de AI losgekoppeld kan worden van de generatie van bewegende beelden in het spel. Dit creëert uiteraard een kleine overhead, maar kan het spel vlotter doen lopen. Het zal er zeker voor zorgen dat het efficiënter kan uitgevoerd worden op hyper-threaded of multicore processoren. Gezien de computerindustrie focust op CPU's met meerdere cores die meerdere threads parallel kunnen uitvoeren, wordt dit steeds belangrijker.

1.2.2 Bewegende beelden

Een ander belangrijk aspect bij games is het visuele. Er moet een vloeiend beeld weergegeven worden, wil men het spel er goed doen uitzien. De theorie rond hoeveel beelden per seconde het menselijk oog minimaal moet zien om iets als een vloeiende beweging te ervaren, is veel ingewikkelder dan meestal wordt aangenomen. Er zijn een paar vuistregels die in de meeste gevallen (blijken te) kloppen:

- Hoe meer frames per seconde (fps), hoe vloeiender het beeld.
- Hoe kleiner het verschil tussen (de beeldinformatie op) de verschillende frames hoe vloeiender de beweging.

Dit is niet in alle gevallen correct, maar voor ons volstaat het om hier zo over te denken. Het is gemakkelijk om de snelheid van de game loop (logische

lus) in frames per second uit te drukken, op die manier is er een maat om mee te vergelijken. Frames per second is dan eigenlijk hoeveel keer per seconde de game state aangepast wordt.

Het aantal fps is dus bij games enerzijds gelimiteerd door de hardware van de computer, maar ook door hoe zwaar de game loop wordt. Bij een actie ondernomen door de speler die meer berekeningen vraagt bijvoorbeeld, zal de loop-iteratie langer duren en dus het aantal zichtbare fps tijdelijk dalen.

1.2.3 Collision detection

Bij games wordt vaak over collision detection gesproken, het detecteren van botsingen tussen twee of meerdere objecten. In elke stap in de game loop, direct nadat een beweging is uitgevoerd door objecten, moet er vóór het hertekenen van het frame eerst gecontroleerd worden of er geen collision is, zodat er eventueel gepast op kan gereageerd worden. Dit kan bijvoorbeeld het tegen elkaar plaatsen zijn van de twee objecten in plaats van 'in' elkaar of een botsing zijn die de objecten beiden de tegengestelde kant opstuurt. Eens deze collision gedetecteerd en opgelost is, kan het frame hertekend worden.

In een tweedimensionale ruimte valt detectie van botsingen goed mee, maar in driedimensionale ruimtes kan collision detection snel geavanceerd worden. Er zijn veel artikels en boeken geschreven rond dit onderwerp en elk van deze artikels en boeken vereist een goede wiskundige kennis. In deze opgave is de collision detection extreem eenvoudig gehouden.

1.3 Entity system framework

Uit ervaring hebben we geleerd dat een klassieke objectgeoriënteerde aanpak niet altijd de beste resultaten oplevert bij het ontwerpen van een game. Abstracties die het ene moment logisch lijken, kunnen later een hindernis vormen bij het toevoegen van nieuwe functionaliteit. Voorbeelden hiervan zijn:

- Tekenen naar het scherm gebeurt door elke klasse een render-methode te laten implementeren, maar dit zorgt ervoor dat dit aspect van het spel verspreid wordt over de volledige codebase, wat het moeilijker onderhoudbaar en uitbreidbaar maakt.

- Generieke functionaliteit wordt geïmplementeerd in klassen waarvan de objecten die dit gebruiken kunnen overerven, bv. een speler erft over van de klasse InputController om te kunnen reageren op input. Maar wat als het gedrag at runtime plotseling moet veranderen op basis van de toestand? Bv. de speler dient tijdelijk te worden bestuurd door de AI voor een bepaalde cutscene (animatiefilmpje tussenin).

Bovenstaande problematiek wordt ook erkend in de game industrie en één van de voorgestelde oplossingen is het gebruik van een entity system framework. Bij deze datagedreven aanpak worden de verschillende game-objecten niet meer voorgesteld als afzonderlijke klassen, maar als generieke entities die samengesteld worden aan de hand van een bepaald aantal components. Deze components bevatten heel specifieke data die kunnen gekoppeld worden aan een game-object (bv. de positie).

Entities en hun bijhorende components zijn dus pure datacontainers en implementeren geen functionaliteit of gedrag. Dat is de verantwoordelijkheid van de verschillende systems die zullen inwerken op de geregistreerde entities. Voorbeelden hiervan zijn een system om bewegingsvectoren toe te passen en een system dat entities op het scherm kan tekenen.

Voor meer details en het precieze gebruik van een entity system framework, verwijzen we naar volgende artikels:

- `http://www.richardlord.net/blog/what-is-an-entity-framework`
- `http://www.richardlord.net/blog/why-use-an-entity-framework`

2 Overzicht Opgave

De opgave bestaat uit een aantal aspecten waarover we hieronder een korte introductie geven.

2.1 Implementatie entity system framework (Deel 1, in C)

Het doel van het project is om een puzzelspel te ontwikkelen. In dit puzzelspel bevindt de speler zich in een ruimte waar verschillende deuren de toegang tot kamers blokkeren. De bedoeling is de uitgang te bereiken. Er zijn verschillende soorten sloten die kunnen gebruikt worden om de deuren te openen. Sommige deuren openen pas als er meerdere sloten geopend zijn. Er zijn ook verschillende sleutels te vinden, waarbij elke sleutel slechts op bepaalde sloten past. Sleutels kunnen ook weer uit sloten gehaald worden om ergens anders gebruikt te worden. Door in de juiste volgorde sloten te openen kan de uitgang bereikt worden.

Uiteraard is er heel wat meer te vertellen over de spelregels, maar dit korte overzicht geeft alvast een houvast over hoe het spel opgebouwd is. Om het spel te implementeren, moeten de level files gelezen worden, de spelregels geïmplementeerd worden m.b.v. een entity system framework, en moet een geheugenbeheersysteem voor het entity system framework geïmplementeerd worden.

De gedetailleerde opgave omtrent het entity system framework kan je terugvinden in Sectie 5.4.

2.2 Sensoren en LEDs (Deel 1, in C op Raspberry Pi)

Jullie krijgen voor dit project per groep van vier studenten, twee Raspberry Pi 3 B's met Sense HAT ter beschikking. Jullie moeten de sensoren op de Sense HAT uitlezen en deze waarden gebruiken om de kleuren van het spel aan te passen. Communicatie met de sensoren gebeurt via I2C (zie Sectie 4.4). De gedetailleerde opgave omtrent het aanspreken van de sensoren op de Sense HAT kan je terugvinden in Sectie 5.7.1.

Ook de LEDs op de Sense HAT moeten aangestuurd worden. Voor communicatie met de LEDs is een driver aanwezig, waardoor dit via een "framebuffer device" verloopt in plaats van via I2C. De gedetailleerde opgave hieromtrent vind je in Sectie 5.7.4.

2.3 Emulatie sensoren en LEDs (Deel 1, in C op Windows en/of Linux PC)

De sensoren en de LEDs zullen ook op Windows en Linux moeten kunnen gebruikt worden. De sensoren en LEDs zijn op deze platformen uiteraard standaard niet aanwezig, dus die zullen geëmuleerd moeten worden. Er moet een manier voorzien worden waarmee de gebruiker geëmuleerde sensor data kan ingeven. In plaats van de Sense HAT LEDs aan te sturen, zal een BMP afbeelding weergegeven en naar bestand weggeschreven moeten worden. Voor de rest van het spel moet het niet uitmaken of er echte of geëmuleerde sensoren en LEDs zijn.

Verdere informatie over de emulatie van sensoren en LEDs kan je terugvinden in Sectie 5.8 en 6.6.

2.4 Spel Movie en Menu op PC (Deel 2, in C++)

Voor het starten van het spel moet een intro "movie" weergegeven worden. Daarna moet een menu verschijnen waarmee de speler het spel kan starten. Beide vereisen het weergeven van tekst en het toepassen van animaties op die tekst.

Dit moet op een correct gestructureerde, objectgeoriënteerde manier ontwikkeld worden, met gebruik van de nodige designpatronen. In Sectie 6.3 krijgen jullie gedetailleerde uitleg over de vereisten die we aan deze implementatie stellen.

2.5 Spel Menu op Raspberry Pi LEDs (Deel 2, in C++)

Op de Raspberry Pi wordt het menu niet alleen weergegeven op het scherm, maar ook op de LED-grid van de Sense HAT. Uiteraard zijn er op dit scherm van 8x8 pixels minder weergavemogelijkheden. Voor dit deel moet een klein lettertype worden ingelezen en weergegeven, en moeten eenvoudige afzonderlijke animaties gebruikt worden. Meer info in Sectie 6.5

2.6 Integratie IMU (Deel 2, in C++)

Op Raspberry Pi moet de IMU (Inertial Measurement Unit) gebruikt worden om de camera van het spel aan te sturen. Als de Raspberry Pi gedraaid wordt, moet de camera op het scherm meedraaien. Hiervoor wordt een C++ library gebruikt. Meer info in Sectie 6.4

3 Bibliotheken

Een aantal hulpklassen zijn gegeven en maken het project eenvoudiger. Er worden ook enkele bibliotheken gebruikt. Met sommige daarvan moet geen rekening gehouden worden in het project, ze zijn afgeschermd door hulpklassen. Andere bibliotheken moeten in beperkte mate gebruikt worden.

De gebruikte externe bibliotheken zorgen er in hoofdzaak voor dat het eenvoudiger wordt om op een platformonafhankelijke manier een venster met 3D weergave te openen en om de toetsenbord- en muis-input te lezen. Een belangrijk deel van software-ontwikkeling is het integreren en gebruiken van bestaande bibliotheken. Het heeft geen zin om telkens het wiel opnieuw te gaan uitvinden, zeker niet als iemand anders gespecialiseerd is in wielen maken.

Wij leveren zelf ook een bibliotheek aan om grafische weergave te vereenvoudigen (zie sectie 3.1).

De gebruikte externe bibliotheken zijn:

- OpenGL: Dit is eigenlijk geen bibliotheek, maar een interface om gebruik te maken van de grafische kaart. Er is een C versie van deze interface beschikbaar, maar er is ook ondersteuning in andere talen. Deze manier om een grafische kaart aan te sturen is door verschillende bedrijven overeengekomen, waardoor er bijna op elk systeem ondersteuning voor is (van laptops tot smartphones en zelfs browsers). Er zijn verschillende versies van deze interface. Wij maken gebruik van OpenGL ES 2, omdat die door de (beperkte) grafische kaart van de Raspberry Pi ondersteund wordt. Ook bijna alle PC's en laptops ondersteunen deze versie (aangezien ze nieuwere versies ondersteunen, en die backwards compatible zijn).
- SDL: Deze C bibliotheek laat toe om op een platformonafhankelijke manier een venster te openen, de muis- en toetsenbord-input te lezen, en OpenGL te initialiseren. Zie sectie 3.3.
- GLEW: Deze bibliotheek wordt gebruikt om op een eenvoudige wijze OpenGL verder in te stellen.
- glmc: Dit is een minimalistische port van de C++ library "glm". Het

kan handig zijn hier gebruik van te maken. Zie sectie 3.2 voor meer detail.

- RTIMULib: Deze bibliotheek ondersteunt de IMU chip op de Raspberry Pi. Ze verzorgt alle communicatie via I2C met de IMU chip en verwerkt de data tot een betrouwbare stabiele positie. Zie sectie 6.4 voor meer uitleg.

3.1 3D grafische weergave

De meeste functies voor grafische weergave hoeven niet door jullie aangesproken te worden. Ze worden reeds gebruikt in het render system dat door ons gegeven wordt.

De functies om tekst weer te geven zullen jullie wel rechtstreeks moeten gebruiken (in deel 2 van de opgave). Deze functies staan in de header file `graphics/gl_glyph.h`. Het gebruik ervan wordt geïllustreerd in `pi_escape2_part2_main.cpp`. Een kleine toelichting bij de gebruikte functies:

- ```
  void gl_glyph_init(GLGlyph*,
                     Graphics*,
                     char* font_image_filename);
  ```

  Initialiseert de `GLGlyph` struct die gebruikt wordt om met behulp van `gl_glyph_draw` karakters te tekenen. De `font_image` wordt ingelezen en klaargemaakt voor weergave. Deze methode moet slechts 1 keer per font opgeroepen worden.

- ```
  void gl_glyph_draw(GLGlyph* obj,
                     int pos_ltop_x, int pos_ltop_y,
                     int glyph_x, int glyph_y,
                     int glyph_w, int glyph_h,
                     const t_vec4 color);
  ```

  Tekent met behulp van `GLGlyph` een karakter op een positie aangegeven door `pos_ltop_x` en `pos_ltop_y`. De waarden `glyph_x` en `glyph_y` geven de positie van de linkerbovenhoek van de glyph terug in de font image opgeslaan in `GLGlyph`, `glyph_w` en `glyph_h` geven de width en height van de glyph aan. De kleur waarin de glyph moet getekend

worden wordt aangegeven door een vector met vier waarden tussen 0.0 en 1.0 (RGBA, dus rood, groen, blauw en alpha. Alpha 0 staat voor volledig transparant en alpha 1 staat voor niet transparant).

- `void gl_glyph_free(GLGlyph *)`: Vrijgeven gebruikte geheugen.
- `void graphics_begin_draw(Graphics*);`

  Het klaarmaken van het tekenen van een nieuw frame. Dit moet gebeuren bij het begin van elke game loop iteratie.
- `void graphics_end_draw(Graphics*)`: Het eigenlijke tekenen van een frame, op het einde van elke game loop iteratie.

  Alle `gl_glyph_draw` calls moeten tussen `graphics_begin_draw` en `graphics_end_draw` gebeuren.

3.2 glmc library

De glmc bibliotheek definieert datastructuren voor vectoren en matrices, en implementeert operaties erop. Deze bibliotheek, wordt o.a. gebruikt in de (gegeven) bibliotheek voor grafische weergave. Het kan handig zijn om ze ook op andere plaatsen in het project te gebruiken.

glmc is gebaseerd op de C++ glm bibliotheek. Vergeleken met de C++ versie, is de C versie extreem beperkt in functionaliteit en minder handig in gebruik. De syntax van de C++ glm bibliotheek is op zijn beurt gebaseerd op GLSL, de "Shader language" van OpenGL. Vectoren en matrices zijn in deze taal zeer makkelijk te gebruiken. Het is voor het project niet nodig om alle details van GLSL, de C++ glm bibliotheek of glmc te kennen.

De opgave code bevat een voorbeeld van het gebruik van de glmc datatypes en functies, zie `glmc_demo_main.c`.

3.3 SDL

De "Simple DirectMedia Layer" bibliotheek laat toe om op verschillende platformen audio, toetsenbord, muis, joystick en grafische hardware (via OpenGL) op een gelijkaardige manier aan te spreken. We maken binnen

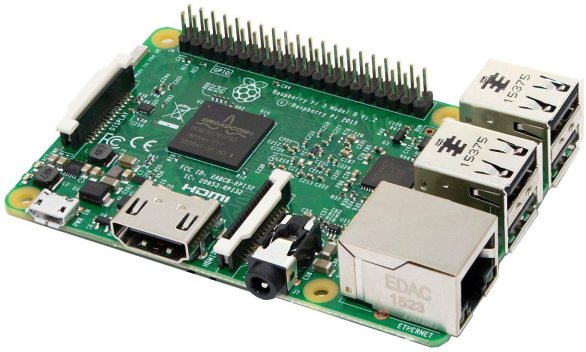
het project gebruik van SDL1.2, niet van SDL2.0. Documentatie hierover vind je op `https://www.libsdl.org/release/SDL-1.2.15/docs/html/`.

Binnen het project moet enkel gebruik worden gemaakt van SDL voor het afhandelen van input komende van muis en toetsenbord. De code om SDL te initialiseren en om een venster te openen met SDL wordt reeds afgehandeld in de gegeven code in `opengl_game_renderer.c`.

De opgave file `input_system.c` bevat demo code voor alle SDL functionaliteit die relevant is binnen het project.

4 Raspberry Pi & Sense HAT

4.1 Raspberry Pi 3



Figuur 1: De Raspberry Pi 3.

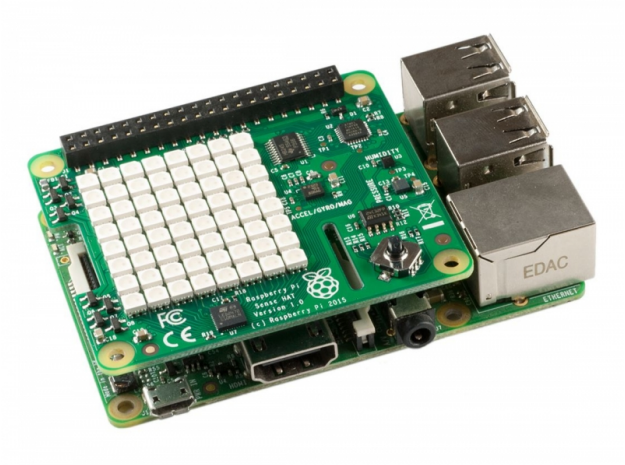
De Raspberry Pi 3 is een single board computer gemaakt voor educatieve doeleinden. Dit bordje heeft naast een moederbord, CPU, GPU en RAM geheugen ook een chip die 802.11n 2,4 GHz WIFI en Bluetooth 4.1 Low Energy aanbiedt. Daarnaast heeft de Pi diverse connectoren zoals een ethernet poort, USB poorten en een HDMI adapter. Meer informatie vind je op de Raspberry Pi website [1].

4.2 Raspbian Linux

Raspbian Linux is een derivaat van Debian Linux die speciaal gemaakt is om op de Raspberry Pi te draaien. Raspbian heeft een hoop handige educatieve tools en wordt steeds verbeterd zoals met de laatste "pixel" update [2]. Raspbian, Debian en Ubuntu zijn alle drie Linux distributies van de Debian familie. Deze lijken dus heel hard op elkaar. Instructies voor Debian en Ubuntu werken meestal ook voor Raspbian. Om te compileren op dit platform kan je dus ook de instructies in Appendix C.2 volgen.

[1] `https://www.raspberrypi.org/`
[2] `https://www.raspberrypi.org/blog/introducing-pixel/`

4.3 Sense HAT



Figuur 2: De Raspberry Pi 3 met Sense HAT.

De Sense HAT [3] is een uitbreidingsbordje voor de Raspberry Pi origineel gemaakt voor de Astro Pi missie [4]. Dit bordje heeft onder andere een RGB LED-scherm en een hoop sensoren. In kader van het Astro Pi project konden kinderen Python programma's schrijven om op een Astro Pi in het ISS te laten runnen. Voor dit project gaan we iets meer low-level en gaan we de Sense HAT aanspreken aan de hand van de Linux Kernel, C en C++.

4.4 I2C

Bij de documentatie op Github vind je een korte intro tot I2C: `https://github.ugent.be/Systeemprogrammeren/public-documentation/blob/master/i2c-intro.md`

[3] `https://www.raspberrypi.org/products/sense-hat/`
[4] `https://www.raspberrypi.org/education/programmes/astro-pi/`

5 Opgave deel 1: Pi Escape spel, sensoren en LEDs (C)

5.1 Algemeen

De C11 standaard wordt gebruikt in dit deel van het project. Dat betekent o.a. dat initialisatie van variabelen niet vooraan in een functie moet gebeuren. Dit is voordelig voor de leesbaarheid van de code.

We raden sterk aan gebruik te maken van de datatypes in stdint.h. Datatypes zoals `uint16_t` hebben een vaste lengte, terwijl het datatype `unsigned int` niet noodzakelijk op elk platform en voor elke compiler dezelfde lengte heeft.

Verder moet de code zowel op Linux als op Windows, als op de Raspberry Pi werken. Zie appendix C omtrent het compileren op deze platformen en appendix B omtrent platform-onafhankelijk programmeren.

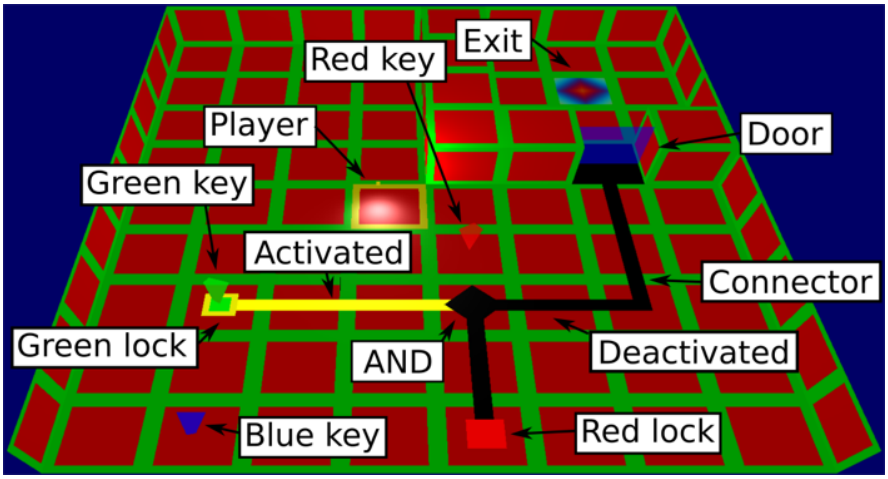
5.2 Spelregels

Pi Escape is een puzzelspel met meerdere levels. De eerste zeven levels vormen een inleiding die de basisconcepten demonstreren, daarna volgen drie puzzellevels waar de speler moet puzzelen. In elk level start de speler op een vaste plaats en moet hij de uitgang van het level bereiken. Er kunnen meerdere uitgangen zijn, in dat geval moet om het even welke uitgang bereikt worden.

Elk level is rechthoekig en bestaat uit rijen en kolommen van cellen. Elk van deze vierkante cellen is even groot en is één van meerdere mogelijke types. Zo zijn er lege cellen waar de speler over kan lopen en muurcellen waar de speler omheen moet. Cellen kunnen vaste objecten bevatten, zoals deuren, verbindingsstukken en sloten. Ze kunnen ook verplaatsbare objecten bevatten, zoals sleutels en de speler zelf.

De uitgangen zijn versperd door deuren die initieel gesloten zijn. Deuren moeten altijd tussen twee muren staan, en zijn via verbindingsstukken verbonden met één of meerdere sloten. Deze sloten kunnen geopend worden

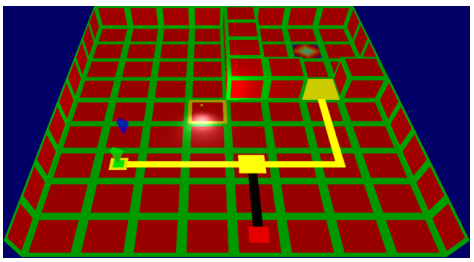
door de sleutels (weergegeven als kristallen in het spel) die in het level te vinden zijn. De speler kan op elk moment slechts één sleutel vastnemen en verplaatsen. De speler kan sleutels uit hun slot halen om ze ergens anders te gebruiken. Een deur zal opengaan van zodra een verbonden verbindingsstuk geactiveerd is. Elke deur kan langs elk van de twee zijden een verbindingsstuk hebben. Slechts één verbindingsstuk moet geactiveerd zijn om dergelijke deur te openen.



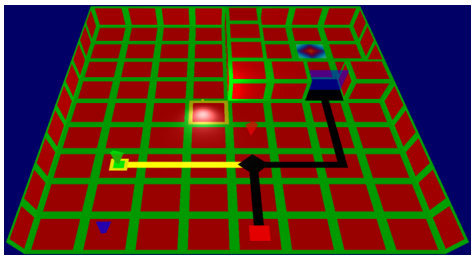
Figuur 3: Spelconcepten

Sloten zijn verbonden met deuren via verbindingsstukken. Je kan een slot activeren door er een passende sleutel in te plaatsen. Het verbindingsstuk verbonden met het slot zal activeren, en het verbindingsstuk verbonden met dat verbindingsstuk zal vervolgens ook activeren. Op deze manier zal je dus een pad aaneengesloten verbindingsstukken activeren, en als dat eindigt bij een deur, zal de deur openen. Als de sleutel weggenomen wordt, zal het verbindingsstuk deactiveren, zal het ganse pad verbindingsstukken deactiveren en zal de eventuele deur weer sluiten. Het activeren van buurverbindingsstukken verloopt veel trager dan het deactiveren, dat bijna onmiddellijk gaat. Op die manier is het onmogelijk dat een speler snel door een deur kan stappen, net voor die gesloten wordt.

Sleutels passen enkel op sloten die dezelfde kleur hebben (groen, rood of blauw). Er zijn echter speciale sleutels en speciale sloten, waarvoor andere regels gelden: een veelkleurige sleutel past op alle sloten en op een wit slot



Figuur 4: Of-verbindingsstuk



Figuur 5: En-verbindingsstuk

past elke kleur sleutel (inclusief de veelkleurige sleutel). Een blauwe sleutel zal dus een blauw of een wit slot activeren, maar een groen slot niet. Een veelkleurige sleutel activeert elk slot.

Verbindingsstukken vormen een pad, waarin geen zijpaden toegelaten zijn. Enkel twee "logische" verbindingsstukken kunnen meer dan twee verbindingsstukken verbinden. Deze logische verbindingsstukken verbinden twee of drie verbindingsstukken (gemakshalve hier de "upstream" verbindingsstukken genoemd), en als de voorwaarde voldaan zijn, dan activeren ze één ander verbindingsstuk (gemakshalve hier het "downstream" verbindingsstuk genoemd). Dat downstream verbindingsstuk is altijd het verbindingsstuk in de richting van de deur. Uiteraard kunnen er ook nog wel een of meerdere andere (al dan niet logische) verbindingsstukken tussen de deur en het downstream verbindingsstuk zitten. Het eerste type logische verbindingsstuk is het "of" verbindingsstuk (zie Figuur 4), dat het downstream verbindingsstuk zal activeren als minimaal een van de upstream verbindingsstukken geactiveerd is. Het tweede type logische verbindingsstuk is het "en" verbindingsstuk (zie Figuur 5), dat het downstream verbindingsstuk slechts zal activeren als alle upstream verbindingsstukken geactiveerd zijn.

5.3 Formaat levels

Er zijn bestanden met levels meegegeven met de opgave. Deze moeten door het spel worden ingelezen. Het bestandsformaat is een eenvoudige tekstuele weergave van het level. Elk karakter komt overeen met een cel in het spel.

Elke regel in een level-bestand komt overeen met een rij cellen in het spel. Regels worden gescheiden door `\n` op Linux systemen, en door `\r\n` op

Windows systemen. Zorg er voor dat beide regeleindes ondersteund worden, zelfs als ze door elkaar gebruikt worden (wat kan gebeuren als je dezelfde levels zowel op Linux als op Windows bewerkt). Volledig lege lijnen (dus ook geen spaties) onderaan en bovenaan het bestand moeten overgeslagen worden. Niet elke rij moet evenveel karakters bevatten als er kolommen zijn in het level. Als een rij korter is dan het aantal kolommen, wordt deze aangevuld met lege cellen.

Het aantal rijen in het level komt dus overeen met het aantal regels in het level-bestand, verminderd met het aantal lege regels in het begin en op het einde van het bestand. Het aantal kolommen in het level komt overeen met het aantal karakters op de langste regel in het level-bestand. Hieronder een overzicht van welke karakters met welke cel types overeenkomen:

Karakter(s)	**Cel type**
spatie	Lege cel
`= - .`	Cel met verbindingsstuk
`* X x`	Muur
`S s`	Startpositie speler
`E e`	Uitgang van het level
`D d #`	Deur
`&`	En-verbindingsstuk
`|`	Of-verbindingsstuk
`A B C O`	Slot met 1 van de 3 kleuren (A B C), of geen kleur (O)
`A b c o`	Sleutel met 1 van de 3 kleuren (a b c), of veelkleurig (o)

Opgave: Implementeer in `levelloader.h` en `levelloader.c` het inlezen van de het level-bestanden in C. Sla het resultaat op in een struct met als naam **Level**.

5.4 Entity system framework

5.4.1 Algemeen

Algemene informatie over entity system frameworks is reeds gegeven in sectie 1.3. In het project dienen jullie het spel binnen een dergelijk framework te implementeren. Een entity system framework maakt gebruik van enkele

basis concepten, waarvan we de C implementaties binnen het project kort één per één overlopen.

Component Een component is een heel eenvoudige data-container, die data ("attributen") bevat over een entity. Sommige components bevatten zelfs geen attributen, hun al dan niet aanwezigheid bij een entiteit kan op zich al voldoende informatie zijn.

In de C implementatie van het project is er voor elk type component een specifieke C struct, en een bijbehorende waarde in de enum `ComponentId`. Deze `ComponentId` wordt gebruikt om het type component bij te houden, aangezien er in C geen manier is om deze informatie rechtstreeks bij te houden. In talen met introspectie/reflectie kan dit wel. Zo kan je in Java het object "Class" gebruiken om via reflectie types te onderscheiden.

Entity Een entity stelt een "ding" voor in het spel. Conceptueel is een entity een samenstelling van components. In de C implementatie bestaan entities enkel uit een unieke ID. Het geheugenbeheer systeem (Zie sectie 5.5) zal voor elke entity ID bijhouden welke specifieke components er bij horen.

System Een system stelt een bepaald functioneel aspect van het spel voor. Bijvoorbeeld de rendering, de movement, etc. Het system bevat de code die de specifieke functionaliteit implementeert. Alle systems hebben een `system_<name>_update` functie, die door de engine in elke game loop iteratie aangeroepen wordt. In het algemeen zal elk system in elke iteratie van de game-loop specifieke entities zoeken en aanpassen.

Engine De Engine is het centrale onderdeel van het framework. Hier komt alles functioneel samen. Het bestaat uit een struct en enkele functies. Engine heeft een `engine_update` functie die zal opgeroepen worden voor elke iteratie van de game loop. In deze functie wordt elk system aangeroepen. Systems zullen via het geheugenbeheer systeem entity ID's en bijbehorende components opvragen. De data van het geheugenbeheer systeem wordt in de engine struct bijgehouden.

Context Dit is een hulp struct waarin data wordt bijgehouden die niet tot één specifiek deel van het spel behoort, maar tot het gehele spel. Een voorbeeld van data die in Context kan worden bijgehouden is de huidige tijd.

Conceptueel kan je context zien als een enkele entity die het spel zelf voorstelt, en die een unieke component met globale speldata bevat. Geen enkele andere entity moet ooit een component van dat type bevatten, en er moeten nooit meerdere van deze entities zijn. Gemakshalve wordt Context daarom buiten het systeem van de components en entities gehouden. Dit is echter niet strikt noodzakelijk. De Context wordt bijgehouden in de Engine struct.

Assembly Een concept dat vaak bij een entity system framework voorkomt is de "Assembly" of "Assemblage". Deze Assembly biedt functies aan die typische soorten entities en al hun components aanmaken. Zo zal een Assembly voor Pi Escape 2 bv een functie `create_lock_entity` en een functie `create_key_entity` bevatten. Een `create_level_entities(Level*)` functie kan ook handig zijn.

In assembly.h en assembly.c is een voorbeeld assembly voorzien, met als enige functie `create_demo_entities`. De entities die hier aangemaakt worden zijn slechts bedoeld ter illustratie van het render system, en zijn slechts een partiële versie van de entities die in het spel gebruikt zullen worden. Het is uiteraard de bedoeling om de Assembly uit te breiden met nuttige functies.

5.4.2 Systems en components in PiEscape

De verschillende gebruikte systems en components hangen af van de functionaliteit in het spel, en van hoe gekozen wordt deze op te splitsen. De keuze van welke systems er zijn en wat ze doen, hangt zeer nauw samen met de keuze van welke components er zijn.

Wij geven verder in dit document enkele mogelijke systems en components mee. Er zijn zeker andere logische opsplitsingen mogelijk. Elke opsplitsing heeft voor en nadelen, en er is niet altijd een duidelijke "ultieme" opsplitsing.

Jullie zijn vrij om een andere opsplitsing in systems en components te gebruiken, zo lang de opsplitsing logisch blijft en geen onnodige complexiteit

toevoegt.

Opgave: Implementeer in C het spel PiEscape zoals beschreven in Sectie 5.2 en maak daarbij gebruik van de meegeven entity system framework functies. Bepaal zelf een logische opsplitsing in components en systems.

De meeste systems houden zelf geen data bij, alle data zit in de context en in de components. Slechts in uitzonderlijke gevallen moet een system wel data bijhouden, bijvoorbeeld als het met output en input werkt.

Merk op dat niet alle components noodzakelijk datavelden bevatten. Sommige components hebben enkel betekenis door hun al dan niet aanwezig zijn.

Ter inspiratie geven we een kort overzicht van de systems en components die wij gebruiken.

Systems:

- Render system: Zorgt voor het renderen van alles wat een ArtComponent en CellLocationComponent heeft.
- Animation system: Beheert de timing van de animaties, en verwijdert de animaties wanneer ze voltooid zijn.
- Camera system: Bepaalt hoe de camera gepositioneerd is, en zorgt dat de camera niet te snel van positie verandert.
- Action system: Zoekt naar entities met een ItemActionComponent component, en voert de actie uit indien mogelijk (oprapen, neerleggen, of verwisselen van de sleutel). Er wordt hiervoor gekeken of op dezelfde locatie (GridLocationComponent) als een entity met een ContainerComponent en een ItemActionComponent een andere entity is die een ItemComponent heeft. De InContainerComponent wordt dan aangepast (verwijderd of toegevoegd).
- Activation system: Propageert de activatie van connectors. Er wordt een tijd bijgehouden om deze activatie stapsgewijs te laten verlopen.
- Container system: Zorgt ervoor dat items (= entities met een InContainerComponent) zich samen met de container waarin ze zich bevinden

bewegen. Hiervoor wordt de GridLocationComponent aangepast wanneer nodig, en wordt er met animaties rekening gehouden.

- EndLevel system: Gaat na of de speler de uitgang heeft bereikt (speler heeft zelfde posities als entity met een ExitComponent). Indien de uitgang bereikt is, start de exit animatie. Na deze animatie wordt de context aangepast zodat het level eindigt.
- Lock system: Kijkt of de correcte sleutel zich op dezelfde positie als een slot bevind, of niet, en activeert of deactiveert het slot.
- Move system: Zoekt entities met een MoveActionComponent, en voert de beweging uit indien mogelijk. Hierbij wordt rekening gehouden met animaties, en met de MoveHistoryComponent die de vorige beweging bijhoudt. Als twee pijltjestoetsen tegelijk ingedrukt worden, zal elke richting afwisselend gekozen worden om zo diagonaal te bewegen en langs muren te bewegen tot de eerste opening.
- Input system: Afhandelen van toetsenbord en muis input via SDL. Dit voegt MoveActionComponent en ItemActionComponent toe wanneer nodig.
- Orientation system: Afhandelen van IMU input (deel 2 van de opgave).

Components:

- CameraLookAtComponent: Focus punt van de camera
- CameraLookFromComponent: Positie van waaruit camera kijkt. Deze component bevat zowel de afstand en hoeken van de camera relatief tegenover de CameraLookAtComponent, als de overeenkomstige absolute positie van de camera.
- MoveActionComponent: De intentie van een entity om te bewegen. Er kunnen meerdere richtingen tegelijk bijgehouden worden (als de speler twee pijltjestoetsen ingedrukt houdt).
- GridLocationComponent: De locatie van een entity in het spel, in "grid" coördinaten
- OneTimeAnimationComponent: De weergave van een éénmalige animatie
- MoveAnimationComponent: De weergave van een bewegingsanimatie

- BlockingComponent: Het al dan niet blokkeren van spelers en objecten, als de entity actief of inactief is
- ItemComponent: Een opraapbaar object in de wereld, met een type en kleur
- InContainerComponent: De container waarin een entity zich bevindt
- ActivationComponent: Het activeren of deactiveren van een entity
- ActivatableComponent: Het al dan niet actief zijn van een entity
- ConnectionsComponent: De verbinding van een entity met andere entities. Dit houdt de "upstream" en "downstream" entity ID's bij.
- LockComponent: Een slot
- ConnectorLogicComponent: Extra vereiste i.v.m. het activeren van geconnecteerde entities (and/or logica)
- MoveHistoryComponent: De vorige beweging van de entity
- ItemActionComponent: De intentie van een entity om een actie uit te voeren.
- DirectionComponent: Een richting (deuren en connectors hebben een richting)
- WallArtComponent: De weergave van vloeren en/of muren. Render system zal voor elke entity die deze component, en een `GridLocationComponent` heeft, muren en/of vloeren tekenen (zoals bepaald door de datavelden van deze component).
- ArtComponent: De weergave van een 3D object op het scherm (behalve muren en vloeren, die WallArtComponent gebruiken). Render system zal voor elke entity die deze component, en een `GridLocationComponent` heeft, een 3D object renderen in het spel. Het type object is bepaald door de datavelden van deze component. Andere eigenschappen van het 3D object (kleur, richting, etc), worden bepaald door andere componenten van dezelfde entity (bekijk de gegeven Render system code voor details).
- ContainerComponent: De entity is een container waarin andere entities kunnen zitten. Dit is een component die geen data velden bijhoudt.

- WalkComponent: De entity is een locatie die toegankelijk is voor spelers en objecten. Dit is een component die geen data velden bijhoudt.
- InputReceiverComponent: De entity ontvangt input van muis en toetsenbord. Dit is een component die geen data velden bijhoudt.
- ExitComponent: Het level is ten einde als de speler op dezelfde locatie als deze entity staat. Dit is een component die geen data velden bijhoudt.

5.4.3 Details Camera, Animation en Render systems in PiEscape

Ter illustratie geven we hier nog meer details voor enkele systems en components. Zoals reeds vermeld, zijn jullie totaal vrij om dit op een andere manier te implementeren.

Camera system In onze implementatie doet het camera system een tussenberekening voor het render system.

Het camera system bepaalt naar welk punt de camera kijkt, en vanaf welk punt de camera momenteel kijkt. Deze info wordt via de CameraLookFromComponent en CameraLookAtComponent doorgegeven aan het render system (beide components bevatten hiervoor een `t_vec3`).

De nieuw positie naar waar gekeken wordt, wordt bepaald aan de hand van de huidige positie (GridLocationComponent) van de entity die de CameraLookAtComponent bevat. Dit is in praktijk de speler entity, maar voor het camera system maakt dit niet uit.

De positie van waar gekeken wordt, wordt bepaald aan de hand van de muis/IMU input en de positie naar waar gekeken word. De muis en IMU input word verwerkt in andere systems, die de afstand, en XY/Z locatie aanpassen in de CameraLookFromComponent (en dus niet rechtstreeks de camera positie).

Het camera system zal bij wijzigingen de camera in kleine stappen bewegen naar de nieuwe situatie. Als dit niet gebeurt, zal de camera in schokken bewegen in de plaats van vloeiend. Om dit te implementeren kan je eventueel

extra info bijhouden in één of beide camera components (maar dat is niet noodzakelijk).

De components gebruikt door het camera system kunnen er zo uit zien:

```
typedef struct CameraLookFromComponent {
    t_vec3 pos;

    float distance;
    float XYdegees;
    float Zdegrees;
} CameraLookFromComponent;

typedef struct CameraLookAtComponent {
    t_vec3 pos;

    uint64_t last_update;
} CameraLookAtComponent;
```

Opmerking: Het datatypes `t_vec3` wordt door de glmc bibliotheek gedefinieerd, zie sectie 3.2.

Animation system Wanneer een beweging uitgevoerd kan worden (dus na een controle of er geen muur of deur in de weg zit), zal die informatie in een MoveAnimationComponent opgeslaan worden en aan de entity die beweegt gehangen worden.

Het animation system zoekt alle entities met een OneTimeAnimationComponent. De "positie" van de animatie wordt bijgehouden als een positie tussen 0 en 1. Het animation system past elke keer het opgeroepen wordt deze positie aan. Als de animatie voltooit is, wordt de OneTimeAnimationComponent verwijderd.

Het animation system leest ook alle entities met een MoveAnimationComponent, en maakt gebruik van de huidige tijd om de animatie aan te passen. Wij kozen ervoor om de CellLocationComponent slechts eenmaal aan te passen, in het move system, voor het toevoegen van de MoveAnimationComponent. De tussentijdse positie van de bewegende entity wordt dan in de MoveAnimationComponent opgeslaan. Als de beweging voltooid is, wordt de MoveAnimationComponent verwijderd.

Render system Het render system zal voor elk frame de rendering voor zich nemen. Hiervoor worden alle entities bekeken die een ArtComponent bezitten. De positie van de weer te geven entity zit in de CellLocationComponent van die entity. Er wordt ook gekeken of er een MoveAnimationComponent aanwezig is, en indien dat zo is, wordt de positie die die daarin zit gebruikt.

Verder zal het render system de camera components opzoeken en de info hieruit gebruiken. Ook de achtergrondkleur word gezocht en aangepast.

5.5 Geheugenbeheer voor het Entity system framework

5.5.1 Default geheugenbeheer

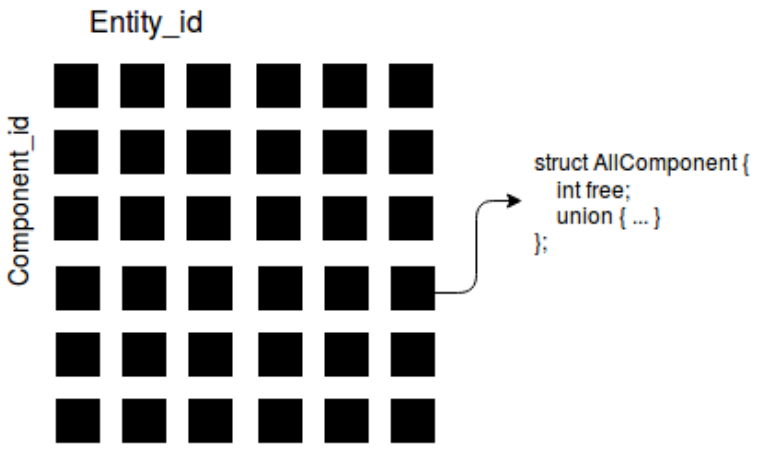
Het meegegeven geheugenbeheer is terug te vinden in `es/es_memory_manager`, deze heeft een struct `ESMemory` die elke engine nodig heeft. Deze struct is de "boekhouding"van het spel. Voor elke entity moet er bijgehouden worden welke componenten deze heeft en hoe deze data bereikbaar is. Naast deze struct heeft het bestand dan ook functies zoals `has_component` en `get_component` die de basisoperaties implementeren op de datastructuur.

Dit geheugenbeheer is zeer eenvoudig gehouden en maakt gebruik van een twee-dimensionale array waar de eerste en tweede index respectievelijk de components en entities aangeeft. Een schematische weergave wordt gegeven in Figuur 6. Het aanmaken van een entity houdt niets meer in dan een teller verhogen en deze terug te geven.

Het zoeken binnen dit beheersysteem is geïmplementeerd in `es/es_memory_manager_collections`. Deze bevat onder andere functies voor het zoeken naar entities die verscheidene components hebben.

5.5.2 Testen geheugenbeheer

Het bestand pi_escape2_test_main.c bevat alle testen voor het geheugenbeheer. Deze moeten worden aangevuld, om zo de volledige functionaliteit te testen.



Figuur 6: Weergave van het default geheugenbeheer.

Voor het schrijven van testen wordt dikwijls van een test framework gebruik gemaakt, zeker in talen zoals Java. Ook voor C en C++ bestaan hiervoor frameworks. Als kennismaking met het schrijven van testen, maken we voor deze opgave gebruik van een minimalistisch test framework, gebaseerd op preprocessor macro's.

In pi_escape2_test_main.c is ter illustratie reeds een test gedefinieerd. Elke test is een afzonderlijke functie. Het return type van de functie is in dit framework steeds `char*`: de functie zal een pointer naar een foutboodschap teruggeven als een test faalt. Als de test succesvol is, wordt de NULL pointer teruggegeven. Daarom staat er "return NULL;" op het einde van elke test functie. Elke test functie zal de waarheid van een of meerdere statements testen. Hiervoor moeten de testfuncties de functie `mu_assert` [5] gebruiken. Deze veronderstelt dat de meegegeven uitdrukking waar is en faalt als dit niet het geval is.

[5] `mu_assert` en `mu_run_test` zijn preprocessor macro's. Dat betekent dat het geen functies zijn die opgeroepen worden, maar dat er code gegenereerd en ingevoegd wordt waar `mu_assert` en `mu_run_test` gebruikt worden. Deze gegenereerde code bevat o.a. een "return", die gebruikt wordt als een assert of test faalt. Daardoor wordt de "return 0" onderaan de test functies en de `test_all()` functie dus enkel opgeroepen als de test succesvol is. Merk op dat het gebruik van macro's op deze manier zeer krachtig is, maar het ook moeilijker maakt om de code snel te begrijpen.

Een goede test zal typisch slechts één functie testen en slechts één bepaalde input waarde voor die functie testen (dit is het "unit test" principe). Uiteraard is het voor de leesbaarheid soms beter om van dit basisprincipe af te wijken, maar het is goed om elke test zo eenvoudig en minimalistisch mogelijk te houden. Over testen valt veel meer te zeggen, het is een zeer belangrijk deel van hedendaagse software ontwikkeling.

De functie `all_tests()` roept alle test functies op via de functie `mu_run_test` en geeft daarna ook NULL terug. Voor het uitvoeren van alle testen wordt een afzonderlijke main functie gebruikt, die de functie `all_tests()` oproept.

Het gebruikte test framework zal stoppen van zodra een test faalt. Complexere frameworks zullen steeds alle testen doen, en zullen ook typisch complexere "assert" functies voorzien. Een test toevoegen kan door een functie die een `char*` terug geeft toe te voegen, en die in `all_tests()` op te roepen via `mu_run_test`.

Opgave: Schrijf voor elke functionaliteit van het geheugenbeheer een test.

5.5.3 Optimalisatie van geheugenbeheer

Het gegeven geheugenbeheersysteem is niet geoptimaliseerd op performantie. Het is statisch en vele operaties zijn inefficiënt om uit te voeren. Er zijn dus heel wat mogelijkheden om het systeem te verbeteren.

Er moet gefocust worden op twee verschillende optimalisaties, het verlagen van gebruikte CPU-tijd of het verlagen van het geheugengebruik. Aanpassen van de ene heeft potentieel een negatieve invloed op de andere. Een balans moet dus gevonden worden zodat beide acceptabel zijn. Een extra datastructuur bijhouden om een veel voorkomende operatie sneller te maken kan het systeem een stuk sneller maken met als trade-off een kleine vergroting in geheugengebruik.

De testen die in de vorige sectie geschreven werden, zullen zeer nuttig zijn om te testen of het geheugen beheer nog correct werkt na de optimalisatie.

Opgave: Implementeer in C een beter geheugenbeheersysteem voor het entity system framework.

5.5.4 Benchmarken van geheugenbeheer

De performantie van het geheugenbeheersysteem kunnen we meten aan de hand van een benchmark. Om een eenvoudige benchmarking uit te voeren die enkel rekening houdt met gebruikte CPU-tijd en het totale geheugengebruik zullen we alle calls naar het geheugenbeheer, opvangen en wegschrijven naar een bestand. Met dit bestand kunnen we alle calls opnieuw afspelen en zo krijgen we een ruwe schatting van de performantie van een systeem. Door hetzelfde bestand te gebruiken bij verschillende optimalisaties kunnen we deze op een consistente manier evaluëren.

Er zijn twee aanpassingen vereist om deze benchmark te implementeren:

- Zorg dat het spel uitgevoerd kan worden in een "benchmark"mode. Tijdens het spelen worden alle calls naar het geheugenbeheer weggeschreven naar een apart "benchmark bestand".
- Maak een nieuwe main die een benchmark uitvoert van het geheugenbeheersysteem. Deze meet de CPU-tijd en het geheugengebruik. De main neemt als argument de naam van een "benchmark bestand", leest het bestand in en voert alle instructies uit.

Opgave: Meet de performantie van het gegeven, en het geoptimaliseerde geheugenbeheersysteem. Meet zowel het geheugengebruik als het CPU-gebruik.

5.6 Raspberry Pi joystick

De mini joystick op de Raspberry Pi moet gebruikt kunnen worden om het spel te bedienen. Merk op dat de Raspberry Pi joystick niet door het systeem als een joystick gezien wordt, maar als de pijltjestoetsen op het toetsenbord. Zorg er dus voor dat het spel met de pijltjestoetsen bestuurd kan worden, dan zal de joystick automatisch ook werken.

5.7 Sense HAT input/output

5.7.1 Sensor input via I2C: temperatuur, luchtdruk en luchtvochtigheid

De Sense HAT bevat twee omgevingssensoren die aanspreekbaar zijn door middel van het I2C protocol: HTS221 en LPS25H, respectievelijk een luchtvochtigheidssensor en een luchtdruksensor. Beide sensoren kunnen ook de temperatuur meten. Heel veel componenten in computers communiceren via I2C. Voor de meeste componenten wordt er echter een kernel space driver geschreven die de I2C communicatie op zich neemt. Die driver stelt dan een API beschikbaar die door user space programma's gebruikt kan worden om de component eenvoudig aan te spreken. Op de Raspberry Pi gaan we de I2C devices echter aanspreken vanuit user space door middel van de Linux kernel I2C adapter. De I2C commandline tutorial [6] gidst je door het aanspreken van de druksensor aan de hand van commandline tools. Deze commandolijn tools zullen goed van pas komen bij het debuggen en controleren van je code.

Voor dit project spreken we de Linux kernel I2C adapter aan vanuit C. Een groot deel van de complexiteit van het I2C protocol wordt zo verborgen door de kernel. De I2C adapter van de linux kernel doet zich voor als een file in `/dev/i2c-*`. Door te schrijven naar en te lezen van die file kan je communiceren met een I2C device. Aanduiden met welk I2C device je precies wil praten kan door middel van ioctl operaties. Een ioctl operatie is een extra operatie die je, naast lezen en schrijven, kan uitvoeren op een file. Wat je precies kan doen met een ioctl operatie hangt af van de file zelf. Op een file die een printer voorstelt kan je bv. ioctl gebruiken om het lettertype aan te passen. Op de I2C adapter file gebruik je ioctl om aan te geven naar welk I2C device je wilt schrijven. In de I2C C tutorial [7] vind je meer uitleg over hoe je deze adapter kan aanspreken vanuit C code.

In de kernel documentatie wordt gepraat over SMBus. Dit is een vereenvoudiging van de I2C standaard ontwikkeld door Intel. Het is een subset van de I2C standaard waardoor alle SMBus devices in essentie compatibel zijn

[6] `https://github.ugent.be/Systeemprogrammeren/public-documentation/blob/master/i2c-commandline-tutorial.md`

[7] `https://github.ugent.be/Systeemprogrammeren/public-documentation/blob/master/i2c-c-tutorial.md`

met I2C, maar niet omgekeerd. Voor dit project maken we geen gebruik van de SMBus functies omdat onze sensoren niet expliciet SMBus compatibel zijn. De eerste opgave is dus om I2C functies te schrijven die de functionaliteit van de SMBus functies implementeren (zie opgave deel "generieke I2C functies"). Daarna schrijf je functies die de sensoren initialiseren, de sensorwaarden opvragen, en de gemeten druk, luchtvochtigheid en temperatuur opvragen. Vervolgens kunnen deze sensoren periodiek worden bevraagd en kan hun waarde worden uitgestuurd via een system (zie 5.7.5).

Nuttige links: Op de pinout van de Sense HAT zie je welke I2C adressen de sensoren hebben.

- Sense HAT pinout [8]

Op Datasheets vind je alle informatie over de specifieke sensoren. Welke registers hebben ze, waarvoor dienen de registers, en hoe initialiseer je de sensor.

- LPS25H Datasheet [9]
- HTS221 Datasheet [10]
- Interpreteren van HTS221 meetwaarden [11]

Commandolijn tools om I2C te debuggen:

- I2C commandline tutorial [12]

Documentatie over de I2C devices aanspreken vanuit C code.

[8] http://pinout.xyz/pinout/sense_hat

[9] http://www.st.com/content/ccc/resource/technical/document/datasheet/58/d2/33/a4/42/89/42/0b/DM00066332.pdf/files/DM00066332.pdf/jcr:content/translations/en.DM00066332.pdf

[10] http://www.st.com/content/ccc/resource/technical/document/datasheet/4d/9a/9c/ad/25/07/42/34/DM00116291.pdf/files/DM00116291.pdf/jcr:content/translations/en.DM00116291.pdf

[11] http://www.st.com/content/ccc/resource/technical/document/technical_note/group0/82/97/c8/ab/c6/da/41/ed/DM00208001/files/DM00208001.pdf/jcr:content/translations/en.DM00208001.pdf

[12] https://github.ugent.be/Systeemprogrammeren/public-documentation/blob/master/i2c-commandline-tutorial.md

- I2C C tutorial [13]

5.7.2 Generieke i2c functies

We maken gebruik van volgende functies om de device file te genereren, lezen en naar te schrijven.

- `int i2c_init_adapter(int addr)`: Initialiseer de I2C device file en zet alles klaar om te communiceren met het meegegeven adres. Geeft de file descriptor van de I2C device file terug.
- `int i2c_write_byte_data(int file, uint8_t reg, uint8_t data)`:
  Schrijf meegegeven data naar meegegeven register. Returns -1 indien falen, 0 indien succesvol.
- `int i2c_read_byte_data(int file, uint8_t reg)`: Lees register uit en return de verkregen waarde. Geef -1 terug indien dit faalt.

Tip: Gebruik de commandolijn tools van de I2C commandline tutorial om te controleren of deze functies correct zijn.

5.7.3 Druk, luchtvochtigheid en temperatuur opvragen

Voor het opvragen van sensorwaarden maken we gebruik van volgende functies.

- `int <hts221|lps25h>_init(int frequentie)`: Initialiseer een I2C adapter voor deze sensor en gebruik deze adapter om de sensor te initialiseren. Stel meegegeven frequentie in. In deze functie lees je ook alle calibratieparameters uit. Opmerking: Vervang de waarden tussen "<>" om de juiste functienaam te verkrijgen. Vb: `int lps25h_init()`
- `double <hts221|lps25h>_read_<humidity|pressure|temperature>()`:
  Lees de sensorwaarde uit, vertaal ze aan de hand van de calibratieparameters, en retourneer ze.

[13] https://github.ugent.be/Systeemprogrammeren/public-documentation/blob/master/i2c-commandline-tutorial.md

Tip1: Na het initialiseren van de sensoren kan het even duren voor de sensor de eerste meting uitvoert. Als je tijdens deze periode de meetwaarde van de sensor opvraagt krijg je een foutief antwoord. Schrijf je code zo dat dit niet kan gebeuren. Het STATUS_REG heeft flags die je hierbij kunnen helpen.

Tip2: Om de efficiëntie te verhogen kan je de berekende waarde opslaan en bij het lezen eerst controleren of de meetwaarde al aangepast is (Zie STATUS_REG). Is deze nog niet aangepast, dan kan je de vorige waarde teruggeven.

Opgave:

Implementeer de hierboven beschreven functies.

5.7.4 LED output

De Sense HAT bevat een 8x8 led matrix. Op de Raspberry Pi kan deze aangestuurd worden in de vorm van een Linux frame buffer device, die een abstractie voorziet voor de grafische hardware. Deze devices kunnen worden aangesproken via speciale device nodes die terug te vinden zijn in `/dev/fb*`. Op de Raspberry Pi zal het Sense HAT led matrix device terug te vinden zijn als `/dev/fb1`. Deze devices kunnen benaderd worden via standaard IO operaties om informatie te lezen of te schrijven. Verder kan via ioctl calls allerlei informatie over de hardware worden opgevraagd. Voorbeelden hiervan zijn de resolutie van de framebuffer, de kleurendiepte, de structuur van de kleurcodes, etc. Voor meer informatie verwijzen we naar de documentatie van een frame buffer device [14] (zie vooral sectie 2), van de frame buffer device API [15] (zie vooral sectie 3) en van het ioctl commando [16].

In `led/sense_led.h` werd alvast volgende functie gedefinieerd:

```
void display_ledgrid(SPGM_RGBTRIPLE* ledgrid,
                     const char* framebuffer):
```

Deze functie neemt als eerste argument een array van `SPGM_RGBTRIPLEs` (een eenvoudige struct die een RGB kleurencode bijhoudt, gedefinieerd in `util/rgb_triple.h`). In deze array stelt elke `SPGM_RGBTRIPLE`

[14] `https://www.kernel.org/doc/Documentation/fb/framebuffer.txt`
[15] `https://www.kernel.org/doc/Documentation/fb/api.txt`
[16] `https://linux.die.net/man/2/ioctl`

de kleur van een pixel voor. Je mag er hier van uitgaan dat `ledgrid` steeds een 8x8 matrix voorstelt en dus een lengte 64 heeft. Het tweede argument is de locatie van het te gebruiken framebuffer device (bv. `/dev/fb1`.). Deze functie geeft de opgegeven array van pixels weer op het framebuffer device, rekening houdende met de karakteristieken van het gespecificeerde device (resolutie, kleurendiepte, etc.) die worden verkregen via ioctl.

Opgave: implementeer de functie `display_ledgrid` in `led/sense_led.c`

5.7.5 Integratie Sensor en LED in PiEscape

Voor het aansturen en verwerken van de sensoren, en het gebruiken van de LEDs maken we gebruik van enkele systems:

- Een real sensors system dat de Raspberry Pi sensorwaarden zal uitlezen en gebruikmaakt van de functies beschreven in Sectie 5.7.1. Deze zal via het entity system framework de data delen met de andere systems.
- Om deze sensor data te verwerken, wordt er gebruikgemaakt van een process sensors system dat op basis van deze data verschillende kleuren zal genereren. Deze kleuren worden gebruikt om verschillende aspecten van het spel, zoals de achtergrondkleur, aan te passen. Meer uitleg over de kleuren die dienen aangepast te worden is terug te vinden in Sectie 5.9.
- Het LED-grid zal tijdens het spel weergeven welke kleur sleutel de speler vast heeft met behulp van een led system.
  - Geen sleutel: alle LEDs uit.
  - Gewone sleutel: alle LEDs zelfde kleur als sleutel.
  - Veelkleurige sleutel: LEDs geven regenboogpatroon weer (animatie is hier niet vereist).

  Hoe je deze LEDs precies aanstuurt is beschreven in Sectie 5.7.4.

Opgave: Implementeer de volgende systems:

- real sensors system: lees de sensorwaarden uit vanop de Raspberry

Pi en maak de waarden beschikbaar voor andere systems.

- process sensors system: gebruik de sensorwaarden om kleuren te berekenen. Voor elk type sensor (temperature, pressure en humidity) worden kleuren berekend.
- led system: het LED-grid geeft de kleur weer van de vastgehouden sleutel.

5.8 Sensor emulatie

Gezien de volledige applicatie ook getest en uitgevoerd moet kunnen worden op een ander platform dan de Raspberry Pi, waar geen sensoren aanwezig zijn, dient er ook sensoremulatie voorzien te worden die sensorwaarden aanpast aan de hand van gebruikersinput.

Maak gebruik van de SDL library om in het spel toetsencombinaties te voorzien die de verschillende geëmuleerde sensoren zullen aansturen. Een mogelijke aanpak is om een aantal toetsencombinaties te voorzien die het type sensor aanduiden en de toetsen `+` en `-` om de waarden te verhogen of verlagen.

Opgave: Voeg sensor emulatie toe

5.9 Achtergrondkleur

Het kleurenschema van het spel zal tijdens het spelen aangepast worden. We maken hiervoor gebruik van de verschillende sensoren. Elk type sensor (temperatuur, druk en vochtigheid) is dus verantwoordelijk om de kleur van een spelelement aan te passen.

De gegenereerde kleuren volgen een regenboog patroon. De kleinste gemeten waarde is dus rood, en naarmate de waarde hoger wordt verandert de kleur naar oranje, geel, groen, blauw en tenslotte paars voor de hoogste gemeten waarde. Het berekenen van deze kleuren gebeurt in het process sensor system (zie Sectie 5.7.5). De verschillende rgb waarden berekenen we m.b.v.

3 sinusfuncties met een faseverschil van 120 graden.[17].

Op de Raspberry Pi moeten de werkelijke sensorwaarden worden gebruikt. Op andere systemen wordt gebruik gemaakt van de geëmuleerde sensor input (zie Sectie 5.8).

De volgende spel elementen moeten aangepast worden:

- Temperatuursensor: de achtergrondkleur van het spel.
- Druksensor: de kleur van de vloertegels.
- Vochtigheidssensor: de kleur van de muren.

Opgave: Implementeer voor de sensoren een functie die de achtergrondkleur, via een regenboogpatroon, aanpast.

Pas `render_system` aan zodat de juiste kleuren gebruikt worden.

[17] https://krazydad.com/tutorials/makecolors.php

6 Opgave Deel 2: Animaties, Menu en IMU (C++)

6.1 Algemeen

Het is de bedoeling dat het spel uit deel 1 van het project vervolledigd wordt door er een intro en outro animatie, een menu en integratie met de IMU aan toe te voegen.

In dit deel van het project wordt gebruikt gemaakt van de C++14 standaard. Behalve het keyword `auto` is gebruik van alle features van C++14 toegestaan. Let er ook op dat het keyword `const` gebruikt wordt waar het nuttig is, en maak gebruik van "getters" en "setters" ipv directe toegang tot datavelden toe te laten. Beperk het gebruik van het keyword `friend`.

Er zijn ook enkele kleine aanpassingen aan het C gedeelte van het project nodig voor dit deel van de opgave. Uiteraard worden die aanpassingen in C gedaan: deel 1 wordt niet omgezet naar C++. Op sommige plaatsen zullen C en C++ code samen gebruikt moeten worden, zowel C code oproepen uit C++ als C++ code oproepen uit C zullen nodig zijn.

Ook voor dit deel van het project moet de code zowel op Linux als op Windows, als op de Raspberry Pi werken.

6.2 Tekst weergeven: FontManager

Om tekst weer te geven is er nood aan een lettertype of font. In Pi Escape 2 worden verschillende fonts gebruikt: `Arcade`, `Zorque` en `Base`. Elk font bestaat uit twee bestanden: een .png bestand die de karakters zelf bevat (font_image) en een .fnt bestand met de metadata. Deze fontbestanden zijn reeds geïncludeerd in het project in `graphics/`. Via de volgende link vind je documentatie over het .fnt bestandsformaat en hoe je dit moet interpreteren: `http://www.angelcode.com/products/bmfont/doc/file_format.html`

In `graphics/FontManager.h` is een eerste aanzet gegeven. Deze header bevat twee klasses: `GlyphDrawCommand` en `FontManager`.

- `FontManager`: inladen van fonts en weergeven van tekst. Deze manager zal een gegeven tekst omzetten in een verzameling glyphs van een gegeven font. Deze component zal ook het tekenen van deze glyphs starten.
- `GlyphDrawCommand`: bevat alle informatie om een gegeven glyph op het scherm te tekenen (positie in het font, positie op het scherm, kleur, etc.).

Neem de gegeven header-code goed door, zo krijg je een duidelijk beeld van wat er moet geïmplementeerd worden. Om de karakters/glyphs weer te geven op het scherm kan je gebruik maken van de code in `graphics/gl_glyph.h` die verder beschreven wordt in Sectie 3.1.

Opgave: Implementeer de klasses `FontManager` en `GlyphDrawCommand` om de gegeven fontbestanden in te lezen en vervolgens deze fonts te gebruiken voor het weergeven van tekst.

6.3 Movie en Menu

Bij het starten van het spel wordt een intro weergegeven, bij het afsluiten een outro. Een intro en outro noemen we Movie omdat het hier gaat om animatie en weergave zonder interactie. Deze weergave wordt gegenereerd met behulp van OpenGL (het gaat hem in dit geval dus niet over het afspelen van een videobestand). De intro moet minstens de titel van het spel weergeven met een passende animatie. De outro geeft een gepaste boodschap, eveneens met animatie. Maak gebruik van de FontManager om de tekst weer te geven.

Na de intro wordt er een menu weergegeven met de volgende **menu-items:**

- Start Tutorial
- Start Game
- Exit

Deze User Interface (UI) moeten jullie opstellen als Model-View-Controller (MVC), een alombekend en veelgebruikt design patroon. Dit patroon splitst een toepassing op in drie eenheden met verschillende verantwoordelijkheden.

Het Model beheert de data, logica en regels van de applicatie, de View krijgt data van het model en geeft deze weer, en de Controller ontvangt input van de gebruiker en manipuleert het model. De gebruiker ziet de View en gebruikt de Controller, hij zal zelf nooit rechtstreeks het model aanspreken. In `anim/UI.h` is hiervoor de basis meegegeven. Jullie dienen van deze klassen te over te erven en zo een eigen MVC structuur op te bouwen voor het menu. Op het internet zijn vele voorbeelden en toepassingen van MVC te vinden[18].

In `anim/GameUICreator.cpp` vind je code (in commentaar) om het menu en de movies aan te maken. Deze code veronderstelt dat er "Builder" objecten zijn, die gebruikt worden om immutable model objecten aan te maken.

"Immutable" houdt in dat deze objecten na creatie niet kunnen aangepast worden (zorg er dus voor dat dat effectief zo is). Dit biedt allerlei praktische voordelen in verband met het beheer van deze objecten en in het algemeen (bijvoorbeeld leesbaarheid van code, eenvoudigere testen en thread-safety [19]). Hoewel veel van deze voordelen hier minder van toepassing zijn, is het gebruik van immutable objecten waar mogelijk een goede gewoonte. Zeker wanneer toepassingen omvangrijker en complexer worden komen de voordelen tot uiting.

Het Builder patroon[20] laat toe om complexe objecten stapsgewijs op te bouwen en deze dan in afgewerkte vorm te creëeren. In dit geval is het builder object dus een mutable object dat het uiteindelijke immutable object zal aanmaken.

De aangemaakte immutable [21] modellen zijn van de klasses `MenuDefinition` en `MovieDefinition`. Om met het MVC patroon het menu weer te geven zal het nodig zijn om het "menu model" aan te passen. Dat "menu model" is dus een object van een andere klasse dan `MenuDefinition` (het zal waarschijnlijk een `MenuDefinition` object bevatten).

Neem de code in `anim/GameUICreator.cpp` goed door, deze geeft aan wat je zeker moet implementeren voor intro, outro en menu (qua tekst en

[18] `https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm`

[19] `https://en.wikipedia.org/wiki/Thread_safety`

[20] Een eenvoudig voorbeeld van het Builder-patroon kan je hier vinden: `https://jlordiales.me/2012/12/13/the-builder-pattern-in-practice/`

[21] Voor meer info rond Immutable en Mutable objecten zie `https://sidburn.github.io/blog/2017/02/27/mutability-vs-immutability-validation`

animaties). Jullie zijn uiteraard vrij om hierop uit te breiden.

Om bij selectie van een bepaald menu item de overeenkomstige actie uit te voeren zal er informatie terug naar de game loop moeten vloeien. Het Observer patroon [22] biedt hier een oplossing voor. Neem het voorbeeld in de voetnoot door en ga na hoe je dit patroon kan gebruiken om tutorial/game te starten, en het spel te beëindigen.

Opgave:

- Zorg dat er een intro en outro wordt afgespeeld bij respectievelijk het starten en het eindigen van het spel.
- Maak een menu weergave waarmee de speler de tutorial en game kan starten, en het spel kan beëindigen.
- Zorg ervoor dat de code in `anim/GameUICreator.cpp` correct uitvoert en met behulp van het Builder patroon alle immutable objecten aanmaakt.
- Maak gebruik van het MVC patroon om de Movie/Menu functionaliteit uit te werken.
- Gebruik het Observer patroon om bij selectie van een menu-item de overeenkomstige actie uit te voeren.

6.4 Camera via IMU

De camera moet behalve via de muis, ook via de IMU worden bestuurd. Dat betekent dus dat als de Raspberry Pi gekanteld wordt, de positie van de camera op het scherm ook wijzigt.

Om de 3D positie van de Raspberry PI op te vragen, maak je gebruik van de RTIMULib. Merk op dat je best eerst de IMU kalibreert. Deze kalibratie hoeft slechts éénmalig uitgevoerd te worden per Raspberry PI.

RTIMULib is reeds in het project geïncludeerd. Info over RTIMULib

[22] Een simpele implementatie van het observer patroon met moderne C++11 idioma's `https://juanchopanzacpp.wordpress.com/2013/02/24/simple-observer-pattern-implementation-c11/`

vind je op `https://github.com/RPi-Distro/RTIMULib/`. Deze info is extreem beknopt, dus geven we ook mee dat er een zeer relevant voorbeeld staat in de file Linux/RTIMULibDrive/RTIMULibDrive.cpp. Merk in dat voorbeeld op dat "imuData.fusionPose" de gemeten "roll, pitch en yaw" bevat, zoals gemeten door de IMU, in respectievelijk "imuData.fusionPose.x()", "imuData.fusionPose.y()" en "imuData.fusionPose.z()". Deze "roll, pitch en yaw" kunnen gebruikt worden om de positie van de camera in het spel te bepalen.

De library zal aangesproken moeten worden vanuit het entity system framework dat geschreven is in C. De library is echter in C++ geïmplementeerd. Om deze vanuit C te kunnen aanspreken moet er dus een wrapper rond de library geschreven worden.

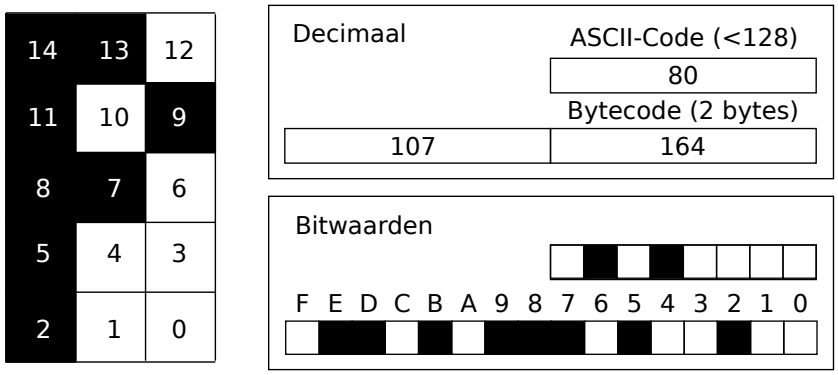
Opgave: Integreer de IMU om de camera aan te sturen via het orientation system. Maak hiervoor gebruik van de RTIMULib.

6.5 LED Menu

Wanneer het programma op de RPI wordt uitgevoerd, moet het menu weergegeven worden op de LED-grid van de Sensehat. In Sectie 5.7.4 kan je zien hoe je deze LEDs kunt aansturen. Door de kleine resolutie van dit LED-grid (8x8) is de functionaliteit van het LED menu natuurlijk beperkt, volgende zaken zijn echter vereist:

- Selecteren menu-item: de tekst van het geselecteerde menu-item moet worden weergegeven op de LEDs. Hiervoor maak je gebruik van het TinyFont dat verder in de tekst beschreven wordt. De tekst past uiteraard niet volledig op het scherm en zal dus automatisch moeten scrollen.
- Activeren menu-item: wanneer een menu-item geactiveerd wordt moet er een passende animatie op het LED-grid worden getoond. Jullie maken zelf uit wat deze animatie inhoudt (bv. het oplichten van de achtergrond).
- Positie in menu: de gebruiker moet zich kunnen oriënteren doorheen het menu en moet dus een indicatie krijgen van zijn positie: hoeveel items zijn er nog voor en na het huidige item.

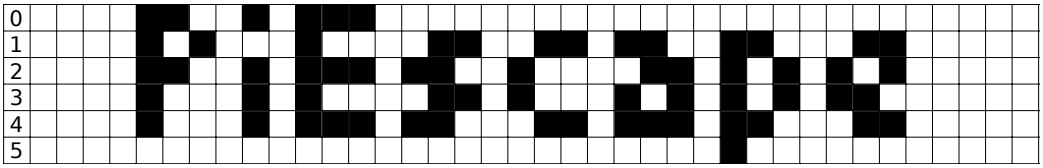
Om de tekst van het menu op het LED-scherm weer te geven maak je gebruik van het TinyFont. Dit is een heel klein, monospace font in een binair formaat. Elk karakter van dit font bestaat uit drie bytes: een byte voor de ASCII-code van het betreffende karakter en twee bytes voor de bitcode waarmee je het karakter pixelgewijs kan weergeven. De karakters staan in ASCII-volgorde in het binaire bestand. De tekenset is echter niet volledig, lees deze karakters dus slim in.



Figuur 7: Weergave hoe een karakter in het TinyFont is geformatteerd. Hier zie je de hoofdletter P, zijn ASCII-code (80) wordt opgeslaan in de eerste byte, de overige twee bytes bevatten de bitcode van het karakter zelf. De zestiende bit geeft aan vanaf welke lijn het karakter getekend wordt (0 of 1).

Elk karakter is maximum drie pixels breed en vijf pixels hoog. De staat van elke pixel wordt weergegeven door de waarde van zijn overeenkomstige bit. Hoe deze bitwaarden op de pixelwaarden worden gemapt is te zien in Figuur 7. Vijftien bits worden gebruikt voor de pixels, het resterende zestiende bit geeft aan vanaf welke rij het karakter getekend dient te worden. Sommige karakters, zoals bijvoorbeeld de 'a' en 'p', dienen een rij lager te worden getekend (zie Figuur 8). Hoewel de karakters dus maximaal vijf pixels hoog zijn is de volledige tekst zes pixels hoog. Als deze zestiende bit gezet is, wordt de eerste rij leeg ingevuld.

Wanneer het programma niet op de RPI wordt uitgevoerd dan wordt er gebruik gemaakt van de LED emulatie besproken in Sectie 6.6.



Figuur 8: PiEscape in het TinyFont weergegeven. Bij de vijf laatste karakters is het zestiende bit gezet.

Opgave: Maak een menu weergave op de LED-grid van de Raspberry Pi met behulp van het binaire TinyFont.

6.6 LED emulatie via BMP

Aangezien de volledige applicatie ook getest en uitgevoerd moet kunnen worden op een ander systeem dan de Raspberry Pi waar geen LED matrix aanwezig is, dient ook de LED matrix geëmuleerd te worden. Hiervoor zullen we gebruikmaken van het BMP bestandsformaat. Een beknopte samenvatting van dit bestandsformaat wordt hier meegegeven, voor meer informatie verwijzen we naar online bronnen [23].

Een BMP bestand bestaat uit twee delen: de eigenlijke bits die de pixels in de afbeelding voorstellen, en een header die het formaat van de bits beschrijft. De header zelf kan gezien worden als twee afzonderlijke delen: de bitmapbestandsheader en de bitmapinfoheader (in deze volgorde). Een korte beschrijving van de structuur van deze headers wordt in wat volgt meegegeven. Het is belangrijk om deze specificaties strikt te volgen om tot een bruikbaar bestand te komen.

- **Bitmapbestandsheader**:
  - bfType (16 bits): definieert het type van het bestand. Wordt ingesteld op 'BM'.
  - bfSize (32 bits): specifieert de grootte van het bestand in bytes. Deze grootte omvat zowel de headers als de eigenlijke bits.
  - bfReserved1 (16 bits): wordt steeds ingesteld op 0.

[23] `https://msdn.microsoft.com/en-us/library/ms969901.aspx`

  - bfReserved2 (16 bits): wordt steeds ingesteld op 0.
  - bfOffBits (32 bits): specifieert de offset van het begin van het bestand tot het begin van de eigenlijke bits. In praktijk komt dit neer op de grootte van de headers.

- **Bitmapinfoheader**:
  - biSize (32 bits): specifieert de grootte van de bitmapinfoheader.
  - biWidth (32 bits): specifieert de breedte van de afbeelding in pixels.
  - biHeight (32 bits): specifieert de hoogte van de afbeelding in pixels.
  - biPlanes (16 bits): wordt steeds op 1 ingesteld.
  - biBitCount (16 bits): bepaalt het aantal bits per pixel. Enkel 1, 4, 8 en 24 bits zijn een geldige waarde.
  - biCompression (32 bits): specifieert het gebruikte type van compressie. Wanneer geen compressie gebruikt wordt, wordt dit veld op 0 ingesteld.
  - biSizeImage (32 bits): specifieert de grootte van de eigenlijke bitmap in bytes. Wanneer het bestand een standaardgrootte geeft (geen compressie), kan dit veld op 0 worden ingesteld.
  - biXPelsPerMeter (32 bits): specifieert applicatie-specifieke dimensies van de bitmap. Wordt meestal op 0 ingesteld.
  - biYPelsPerMeter (32 bits): specifieert applicatie-specifieke dimensies van de bitmap. Wordt meestal op 0 ingesteld.
  - biClrUsed (32 bits): specifieert het aantal kleuren in de kleurentabel. Bij een 24-bits bitmap wordt geen kleurentabel gebruikt en wordt deze waarde op 0 ingesteld.
  - biClrImportant (32 bits): specifieert dat de eerste x kleuren in de kleurentabel de belangrijkste kleuren in de afbeelding zijn. Wanneer alle kleuren even belangrijk zijn (of wanneer er geen kleurentabel is), wordt dit veld op 0 ingesteld.

Na deze headers vinden we de kleurentabel (behalve voor 24-bit bitmaps). Deze tabel bestaat uit een lijst van 32-bit velden (8 bits blauw, 8 bits groen, 8 bits rood, 8 0-bits). De grootte van de kleurentabel hangt af van de waarde van biBitCount (of van biClrUsed indien verschillend van 0) in de bitmapinfoheader.

Vervolgens vinden we de eigenlijke bits van de afbeelding terug. Het formaat van deze bits hangt af van de header, maar volgende algemene regels gelden.

- De pixels worden in de volgorde van onder naar boven, van links naar rechts weergegeven.
- Elke rij in de afbeelding is gealigneerd op 32 bits. Wanneer de pixels in een rij geen veelvoud van 32 bits bevatten, worden padding-bits toegevoegd om tot een veelvoud van 32 bits te komen.

Bij 1-, 4- of 8-bit bitmaps wordt voor elke pixel een 1-, 4- of 8-bit index in de kleurentabel weergegeven. Voor een 24-bit bitmap wordt voor elke pixel 3 bytes opgeslagen, waarvan elke byte een kleur voorstelt (RGB).

Opgave: Implementeer LED-emulatie via het aanmaken van BMP bestanden in `led/fake_led.h`. Maak hierbij gebruik van C++ I/O functies. Wanneer de applicatie niet op de Raspberry Pi wordt uitgevoerd, wordt er automatisch van deze functionaliteit gebruik gemaakt.

7 Praktisch

7.1 Opzetten GitHub repository

Na de introductieles stuurt elke groep 1 email met daarin de GitHub-account namen van alle groepsleden naar sysprog@lists.ugent.be (en naar naar alle groepsleden). Daarna zullen jullie een email krijgen van de voor jullie groep verantwoordelijke assistent.

Deze GitHub accounts zullen dan toegang krijgen tot een private repository met de opgave code. Elke groep krijgt een repository die de groepsleden dan gebruiken als centraal samenwerkingspunt om te clonen, checkout, push/pullen etc. Ook de assistent heeft volledige toegang tot die repository.

7.2 Meetings en contact met assistent

Gedurende het project zullen regelmatig meetings gehouden worden met jullie assistent. Op deze meetings te iGent, Technologiepark-Zwijnaarde 15, 10de verdieping, moet iedereen aanwezig zijn. We gaan er ook van uit dat jullie voorbereid naar de meetings komen: een korte presentatie met stand van zaken, een overzicht van welke teamgenoot wat reeds geïmplementeerd heeft en een planning zijn zeer nuttig. Zorg dat je eventuele problemen of vragen op voorhand geïdentificeerd hebt (en liefst ook reeds aan de begeleider overgemaakt hebt vóór de meeting).

10/10	Introductie deel 1	Gezamenlijke introductiesessie
16/10	Meeting 1	Eerste meeting met assistent, werkverdeling en GitHub
23/10	Meeting 2	Stand van zaken deel 1
05/11	Deadline deel 1	
06/11	Introductie deel 2	Gezamenlijke introductiesessie, algemene feedback deel 1
20/11	Meeting 3	Stand van zaken deel 2
10/12	Deadline deel 2	

Deze meetings zijn bedoeld om jullie te helpen, maak er dus zeker gebruik van om alle kleine en grote vragen die je hebt te stellen. Bereid dus gerust een lijst vragen voor. Het kan erg nuttig zijn om vragen vooraf naar de begeleider te sturen. Naast deze vaste afspraakmomenten kunnen ook ad-hoc afspraken geregeld worden indien nodig, dit hoeft dan ook niet met iedereen te zijn. Uiteraard zijn de assistenten ook steeds bereikbaar via e-mail.

7.3 Deadlines

Het project heeft twee grote deadlines. De eerste voor het afleveren van het C gedeelte, de tweede voor het afleveren van het C++ gedeelte.

- Deadline C (Basis spel)
  - **zondag 5 november 23:59**
  - Laatste versie voor deze tijd wordt automatisch van GitHub gehaald
  - Document op Dropbox indienen (zie 7.5)
- Deadline C++ (volledige spel)
  - **zondag 10 december 23:59**
  - Laatste versie voor deze tijd wordt automatisch van GitHub gehaald
  - Document op Dropbox indienen (zie 7.5)

7.4 Puntenverdeling

Het project geldt voor 5 punten van het examen. Deze 5 punten kunnen als volgt verdiend worden:

- 3 punten op het C gedeelte.
- 2 punten op het C++ gedeelte.

Wanneer er een verschil in inbreng van teamleden wordt vastgesteld, zal er individueel gequoteerd worden.

7.5 Indienen

Alles wordt georganiseerd rond de GitHub repository op github.ugent.be (zie ook cursus). De bedoeling is dat jullie daar gebruik maken van de issue tracker voor problemen en de wiki voor documentatie & informatie.

De assistent kan hierop meevolgen en eventueel zelf antwoorden op issues (graag wel nog altijd een mailtje indien speciale attentie van de assistent nodig is). Ook zal de laatst gepushte commit op de GitHub repository als de finaal ingediende versie gezien worden.

Nog een aantal extra puntjes:

- Het project wordt gemaakt in groepen van vier studenten. Het is verplicht hiervoor een groep op Minerva te kiezen. Indien iemand geen groep vindt (na via Minerva project-forum proberen een groep te vinden), dan vragen we in de eerste week van het project zo spoedig mogelijk contact met ons op te nemen via sysprog@lists.ugent.be
- Hou een kort verslag bij op de GitHub wiki (max. vier bladzijden) met daarin:
  - naam, voornaam en richting van de groepsleden + groepsnummer van Minerva
  - Een extra woordje uitleg bij je ontwerpbeslissingen
  - De taakverdeling: wie heeft wat gedaan?
- De laatste commit op de GitHub repository voor de deadline wordt als final code/state gezien van jullie project.
- Naast de GitHub stuur je ook een document met groepsnummer en de locatie van je GitHub repository (URL) naar de dropbox van Bruno Volckaert op Minerva, en dit voor de deadline voor zowel deel 1 als deel 2 van het project.
- De code zal getest worden op Windows met de compiler van Visual Studio 2017 en op Linux met gcc-6. Zorg dus dat je deze compilers gebruikt voor het maken van de opgave.

Veel succes!

7.6 Minimumvereisten

Bij het indienen van het project vereisen we dat een minimum doel bereikt is. Indien dit niet het geval is, zal een nul-score gegeven worden, en wordt er geen gedetailleerde feedback gegeven.

Voor beide delen van het project moet de code steeds op zowel de Raspberry Pi als op Windows compileren. Er moet ook telkens een verslag aanwezig zijn.

Voor deel 1 van het project is het minimumdoel dat er minstens een level ingelezen en weergegeven wordt (hoe beperkt de functionaliteit van het spel verder ook is).

Voor deel 2 van het project is het minimumdoel dat er minstens een menu is waaruit het spel gestart kan worden.

Deze minimumdoelen zijn makkelijk haalbaar. Merk op dat indien enkel deze minimale doelen gehaald zijn, het project wel zal verbeterd worden, maar de score zeer laag zal zijn.

8 Hints en opmerkingen

- Het is toegestaan (en soms nodig) om opgegeven header files en cpp files aan te passen.
- Begin niet onmiddellijk te programmeren. Lees eerst aandachtig de opgave en denk na hoe je de verschillende problemen zou oplossen (vb. welke klassen ga je gebruiken, wat houden ze bij, etc.).
- Het voordeel van een entity system framework is dat je bepaalde systems kan uitschakelen tijdens het debuggen door ze gewoon even niet aan de Engine toe te voegen.
- Maak in het C++ gedeelte zoveel mogelijk gebruik van de Standard Template Library (STL). Volgende link kan daarbij ook van pas komen: `http://www.cppreference.com/wiki/stl/start`
- Alle opmerkingen bij de oefenset-opgaves gelden ook hier: wees zuinig met geheugen, let op voor dangling pointers, geef alles wat gealloceerd wordt ook weer vrij, controleer of je open bestanden wel correct geopend zijn en gesloten worden, bescherm je header files, etc.
- Als referentiecompiler wordt Microsoft Visual C++ Community Edition 2017 gebruikt. Zorg er dus voor dat je project daarmee compileert en uitvoerbaar is! Behalve het keyword auto is gebruik van C++14 toegestaan.
- Probeer zoveel mogelijk compiler warnings te vermijden; deze duiden meestal op fouten die Visual Studio C++ voor jou zal oplossen, maar die je evengoed kan vermijden door je code aan te passen.
- Vanzelfsprekend zijn er zaken waar deze opgave je vrij in laat; deze mag je zelf kiezen naar eigen inzicht. Verduidelijk in alle geval je broncode met commentaar waar dat nuttig is!

A Eenvoudig memory leaks detecteren in VS

De geheugenchecker in Visual C++ Community Edition kan je gebruiken als volgt. Je includeert in het betreffende bestand (meestal de main) volgende twee headerfiles:

```
#include <stdlib.h>
#include <crtdbg.h>
```

Op de plaats waar je wil nagaan of er geheugen verloren gaat zet je volgende lijn

```
_CrtDumpMemoryLeaks();
```

Als je het programma in debug modus runt, dan krijg je na afloop van je programma melding of er geheugenlekken gevonden zijn, soms kan je uit die melding ook onmiddellijk afleiden waar het lek zit. Let wel, als je variabelen op de stack aanmaakt (=statisch geheugen) in dezelfde functie als waar je `_CrtDumpMemoryLeaks();` statement staat, dan zullen deze variabelen ook gezien worden als memory leaks (aangezien ze nog niet out-of-scope gegaan zijn en dus nog geheugen innemen). Een goede manier om na te gaan of er geheugenlekken zijn is bvb:

```
void actions(...){
//declaraties
//allocaties op heap (via malloc/new/... = dynamisch geheugen
    )
//uitvoeren programma
}
int main(){
actions(...);
_CrtDumpMemoryLeaks();
}
```

Zo ben je zeker enkel melding te krijgen van 'echte' geheugenlekken. Elk geheugenlek heeft een nummer dat je kan zien in de debug?output.

Verder kan je, eens je het nummer van een geheugenlek hebt, ook je programma aanpassen zodat het stopt met uitvoeren op de plaats waar het

geheugen dat het lek veroorzaakt, gealloceerd wordt. Dit kan uitermate handig zijn bij het debuggen! Hiertoe voeg je in het begin van je programma volgende regel toe:

```
_CrtSetBreakAlloc(mem_leak_nr);
```

Naast deze methode voor het vinden van memory leaks, bestaat er een manier om dit proces te stroomlijnen. In plaats van de locatie van het geheugenlek te moeten gebruiken om dan tijdens het opnieuw runnen van je programma een breakpoint te zetten met `_CrtSetBreakAlloc(long)`, kan je met behulp van "Visual Leak Detector for Visual C++" gewoon dubbelklikken op het gevonden lek, om direct naar het bestand en de lijn van de allocatie te gaan. Meer informatie omtrent de installatie en het gebruik van VLD vind je onder "Documentation" op hun Codeplex-pagina: `http://vld.codeplex.com/` (het gebruik onder Visual Studio 2017 is gelijk aan het gebruik onder Visual Studio 2015).

B Platform-onafhankelijk programmeren

Het project moet zowel op Windows, op de Raspberry Pi, en op andere Linux machines uitvoerbaar zijn.

Hiervoor moet rekening gehouden worden met de verschillen tussen deze platformen. Deze appendix geeft een overzicht van enkele manieren om dit te doen.

B.1 Build systems

Build systems zijn verantwoordelijk voor het oproepen van de compiler en linker met de correcte argumenten. Als je een IDE gebruikt is het build system er dikwijls een onderdeel van (dit is zo voor Visual Studio). Enkele build systems kan je via command line oproepen, zoals `make` op Linux. Elk build system houdt in 1 of meerdere files bij hoe het project in elkaar zit. Van deze info wordt gebruikt gemaakt bij het compilen. Een build system zou platformonafhankelijk kunnen zijn, maar in praktijk zit er dikwijls nog een laag boven, die de project files voor het build system genereert. Dat is

bijvoorbeeld typisch het geval met autoconf en automake op Linux (waar je dan als eerste stap `./configure` uitvoert om de makefiles te genereren).

Voor dit project maken we gebruik van "CMake" om de build files te genereren. Op Linux worden door cmake makefiles genegereerd, en op Windows worden uit dezelfde cmake files, de Visual Studio project files gegenereerd. De verschillen tussen platformen worden dus deels door CMake opgevangen.

De file `CmakeLists.txt` bevat alle project info die CMake nodig heeft. Het is mogelijke om in die file bepaalde dingen slechts op bepaalde platformen uit te voeren. Op details ivm CMake gaan we hier niet verder in.

B.2 Preprocessor macro's

In de C en C++ code van het project zijn sommige stukken code afhankelijk van het platform. Om dit op te vangen wordt typisch gebruik gemaakt van preprocessor macro variabelen die enkel op specifieke platformen beschikbaar zijn. De macro variabele `WIN32` is enkel op Windows systemen aanwezig. De macro variabele `LINUX` is enkel op Linux systemen aanwezig. De Cmake files zijn zo aangepast, dat ze de macro variabele `RPI` toevoegen, die dus voor het project enkel op Raspberry Pi aanwezig zal zijn. We kunnen platformspecifieke code toevoegen door de `#ifdef` constructie van de C preprocessor te gebruiken:

```
#ifdef RPI
    engine->orientation_system = init_orientation_system();
#endif
#ifdef _WIN32
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF |
    _CRTDBG_LEAK_CHECK_DF);
#endif
```

B.3 Platformonafhankelijke interfaces

Een manier om code te schrijven die overal hetzelfde is, is om gebruik te maken van platform onafhankelijke interfaces. Deze zijn door een standaardisatie groep gedefinieerd, en worden op de verschillende platformen

geïmplementeerd. Een voorbeeld is POSIX, dat o.a. specifieert hoe in C threads aangemaakt kunnen worden. Een relevant voorbeeld voor het project is OpenGL. Deze interface wordt ondersteund door de makers van grafische chipsets, waardoor verschillende GPU's via dezelfde software kunnen worden gebruikt.

B.4 Platformonafhankelijke bibliotheken

Een aantal bibliotheken is beschikbaar op verschillende platformen, met als doel op elk platform dezelfde code te kunnen gebruiken. De verschillen tussen de platformen worden opgevangen in de implementatie van de bibliotheek. Een voorbeeld in het project is de SDL bibliotheek, die input van muis, toetsenbord en joystick, en output naar scherm en audio aanbiedt op verschillende platformen. Ook voor GUI's zijn platform onafhankelijke bibliotheken beschikbaar, zoals Qt en GTK+.

B.5 Platformonafhankelijkheid via C standard features

De C en C++ standaard bibliotheek bevat verschillende definities en functies om makkelijker platform onafhankelijk te programmeren. De lengte van het `int` datatype is in C afhankelijk van het platform. De `stdint.h` header die in de C standaard gedefinieerd is, heeft verschillende types die wel een platform onafhankelijke lengte hebben, zoals bv `int32_t`. C++11 biedt functionaliteit voor het werken met threads aan, waardoor er geen platform afhankelijke bibliotheken vereist zijn.

B.6 Endian awareness

Om volledig platformonafhankelijk te zijn, is het nodig om endian-aware code te schrijven bij het schrijven en lezen bestanden en bij netwerkcommunicatie. Het omzetten van een `unsigned short` in "host system" order (typisch little endian op x86 hardware) naar big endian, kan met de functie `htons()`. Omzetten van big endian naar "host system" endian kan met `ntohs()`. (Merk op dat de "n" hier voor "network" staat: big endian is de volgorde voor

netwerk communicatie.) Als hier geen aandacht aan besteed wordt, zal op big-endian systemen een bestand weggeschreven worden dat niet compatibel is met little-endian systemen, en vice versa. Merk op dat dat geen probleem is in het project, aangezien het BMP formaat little-endian verwacht, en zowel de processor van de raspberry PI, en Intel x86 en x86-64 processors (of compatibele), gebruiken little-endian. Voor dit project is het dus niet nodig om op endian-awareness te letten.

C Compileren

C.1 Compileren op Windows

Op Windows raden we aan om "Visual Studio 2017 Community Edition" te gebruiken. Je kan deze software downloaden op `https://www.visualstudio.com/downloads/`. Installeer VS 2017 CE en zorg ervoor dat je bij "Workloads" minstens "Desktop development with C++" selecteert.

Op basis van de opgavecode kan automatisch een Visual Studio project gegenereerd worden door gebruik te maken van CMake. Deze open-source software is gratis te downloaden (`https://cmake.org/download/`). Na het installeren gaan jullie als volgt te werk om het project te genereren:

1. Open CMake
2. Selecteer de opgave-map bij "Where is the source code"
3. Selecteer de opgave-map bij "Where to build the binaries"
4. Klik op "Configure"
5. Specifieer "Visual Studio 15 2017" als generator voor dit project
6. Laat "Optional toolset to use" leeg
7. Selecteer "Use default native compilers"
8. Klik op Finish en wacht tot "Configuring done" verschijnt in het statusvenster
9. Klik op "Generate" en wacht tot "Generating done" verschijnt in het statusvenster

Hierdoor heeft CMake enkele projecten en een solution aangemaakt in de opgavemap. Deze solution file (`PiEscape2.sln`) kan je gewoon openen met Visual Studio.

Er zitten zes projecten in de solution (de CMake-specifieke projecten `ALL_BUILD` en `ZERO_CHECK` mogen genegeerd worden):

- `glmc_demo`: Dit bevat `glmc_demo_main.c` met de main functie. Hier vind je voorbeeldcode omtrent het gebruik van de glm library.
- `pi_escape2_part1`: Dit project bevat `pi_escape2_part1_main.c` met de main functie. Dit project start deel 1 van het spel.
- `pi_escape2_part2`: Dit project bevat `pi_escape2_part2_main.cpp` met de main functie. Dit project start deel 2 van het spel.
- `pi_escape2_part1_benchmark`: Dezelfde main als `pi_escape2_part1` maar hier is de benchmark-vlag gezet (vanuit CMakeLists). Zo kan er beslist worden om alle calls naar het geheugenbeheer op te slaan in een benchmark-bestand.
- `pi_escape2_benchmark`: Dit project bevat `benchmarking_main.c` met de main functie. Dit project start een benchmark door het inlezen van gegenereerde benchmark-bestanden.
- `pi_escape2_test`: Dit project bevat `pi_escape2_test_main.c` met de main functie. Dit project start de tests van het geheugenbeheersysteem.

Om een project met een main functie te starten, ga je als volgt te werk:

- Klik met de rechtermuisknop op het project
- Selecteer “Set as StartUp Project”
- Klik vervolgens op de Run knop bovenaan (of gebruik de F5 toets)

De opgave bestanden zullen zonder wijzigingen reeds de (beperkte) functionaliteit hebben, en de testen starten. Uiteraard zal geen van de meegeleverde main functies veel doen, en zal de eerst gestarte test al falen.

Opmerking: Zorg ervoor dat de schaling van Windows (in Control Panel, onder Display - Scale and Layout) op 100% staat. Anders zal het getekende

scherm meegescaled worden en groter dan je eigenlijk resolutie worden getekend.

C.2 Compileren op Linux

Om op Linux te compileren, moeten de vereiste bibliotheken eerst geïnstalleerd worden. Hoe dit moet hangt af van de specifieke Linux distro.

Voor Ubuntu of Debian gebruik je dit commando:

```
sudo apt-get install libsdl1.2-dev libsdl-image1.2-dev libglew-dev
```

Om de projectcode te compileren moeten er ook enkele packages, zoals de compiler, en cmake, geïnstalleerd zijn. Op vele Linux systemen zal dat al het geval zijn. Op Ubuntu kan je ze indien nodig installeren op de volgende manier:

```
sudo apt-get install build-essential pkg-config cmake
```

Op Ubuntu 16 moet gcc 7 geinstalleerd zijn. Dat doe je op volgende manier:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo apt-get install build-essential software-properties-common
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-7 g++-7
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 60
    --slave /usr/bin/g++ g++ /usr/bin/g++-7
sudo update-alternatives --config gcc
```

Het compileren zelf is eenvoudig. Ga naar de map met de opgavecode en voer daar het volgende commando uit:

```
cmake .
```

Dit zal automatisch de bibliotheken detecteren en alles configureren om te kunnen compileren. Om te compileren, voer je dan het volgende commando uit:

```
make -j 4
```

Vervang hier 4 door het aantal cores dat je CPU heeft. De volgende 3 commando's kunnen gebruikt worden om de main functie van deel 1, de testen of de main functie van deel 2 te starten:

```
./pi_escape2_part1
./pi_escape2_part2
./pi_escape2_test
```

C.3 Compileren op Raspberry Pi

Compileren op Raspberry Pi is hetzelfde als op een andere Linux.

Wij hebben echter reeds de nodige software en bibliotheken op de Raspberry Pi geïnstalleerd, waardoor de "apt-get" stappen overgeslaan kunnen worden.