

Inhoudsopgave

Blz. 1	Project Setup	# Overview	[LINK]
Blz. 2		# Handleiding 1/2	[LINK]
Blz. 3		# Handleiding 2/2	[LINK]
Blz. 4	Libzmq Missing	# Oplossing	[LINK]
Blz. 5	1. Game of Life	# Toelichting	[LINK]
Blz. 6		# Set-Up	[LINK]
Blz. 7		# Highlights	[LINK]
Blz. 8	2. Chat Client & Server	# Toelichting	[LINK]
Blz. 9		# Set-Up	[LINK]
Blz. 10		# Highlights	[LINK]
Blz. 11	3. Collision Crisis	# Toelichting	[LINK]
Blz. 12		# Set-Up	[LINK]
Blz. 13		# Highlights	[LINK]
Blz. 14	4. Memory & Cache Optimization	# Toelichting	[LINK]
Blz. 15		# Set-Up	[LINK]
Blz. 16		# Highlights	[LINK]
Blz. 17	5. Multi-Threading	# Toelichting	[LINK]
Blz. 18	SoA & Single-Threaded	# A-Tests	[LINK]
Blz. 19	SoA & Multi-Threaded	# B-Tests	[LINK]
Blz. 20	AoS & Single-Threaded	# C-Tests	[LINK]
Blz. 21	AoS & Multi-Threaded	# D-Tests	[LINK]
Blz. 22	Reflectie	# Persoonlijk	[LINK]

Project Set-Up (#Overview)

Github Link

https://github.com/Weskarr/Low-Level_ComputerScience

Introductie

De eerste week van het blok heb ik de basis overgenomen en werkend gekregen: "SFML-ImGui_StartPoint". Vervolgens de week daarop ben ik gaan kijken hoe ik al mijn opdrachten wilde implementeren over de termijn van de module: alles als losse executables of als enkele?

Redenering

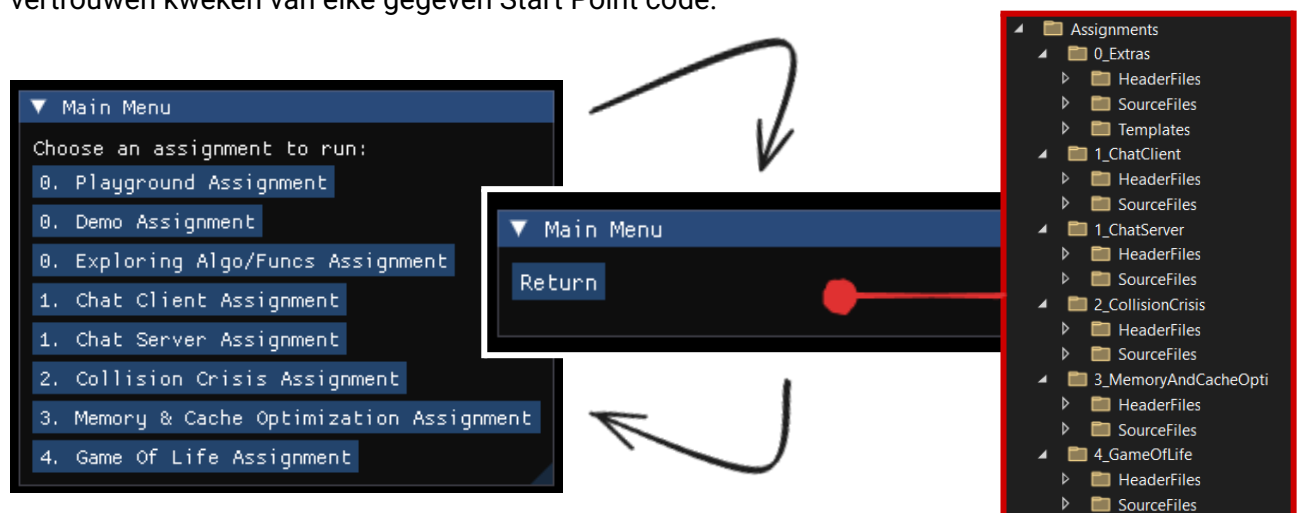
Gezien ik toch ImGui wilde leren, heb ik toen gekozen voor een enkele executable approach en een eigen UI paneeltje gemaakt met een startknop voor elke opdracht. Over het algemeen best ruw en simpel, maar functioneel en vrij uniek.

Set-up

In het kort heeft bijna elke opdracht een eigen folder met de subfolders HeaderFiles & SourceFiles + de assignment.h, die een soort Main extensie is. De enige uitzondering is de Networking opdracht die is opgesplitst naar twee folders: Chat Server & Chat Client en de Multi-Threading opdracht die is verwerkt in Memory & Cache Optimisations. Gezien dat voor dit opdrachten beter uitkwam met het implementeren. Daarnaast heb ik ook een folder met Extras waar de Main, Assignment Class, Templates, ThreadPool en al de basis / globale bestanden in zitten, een soort onderliggende fundering als het ware voor de rest.

Effect

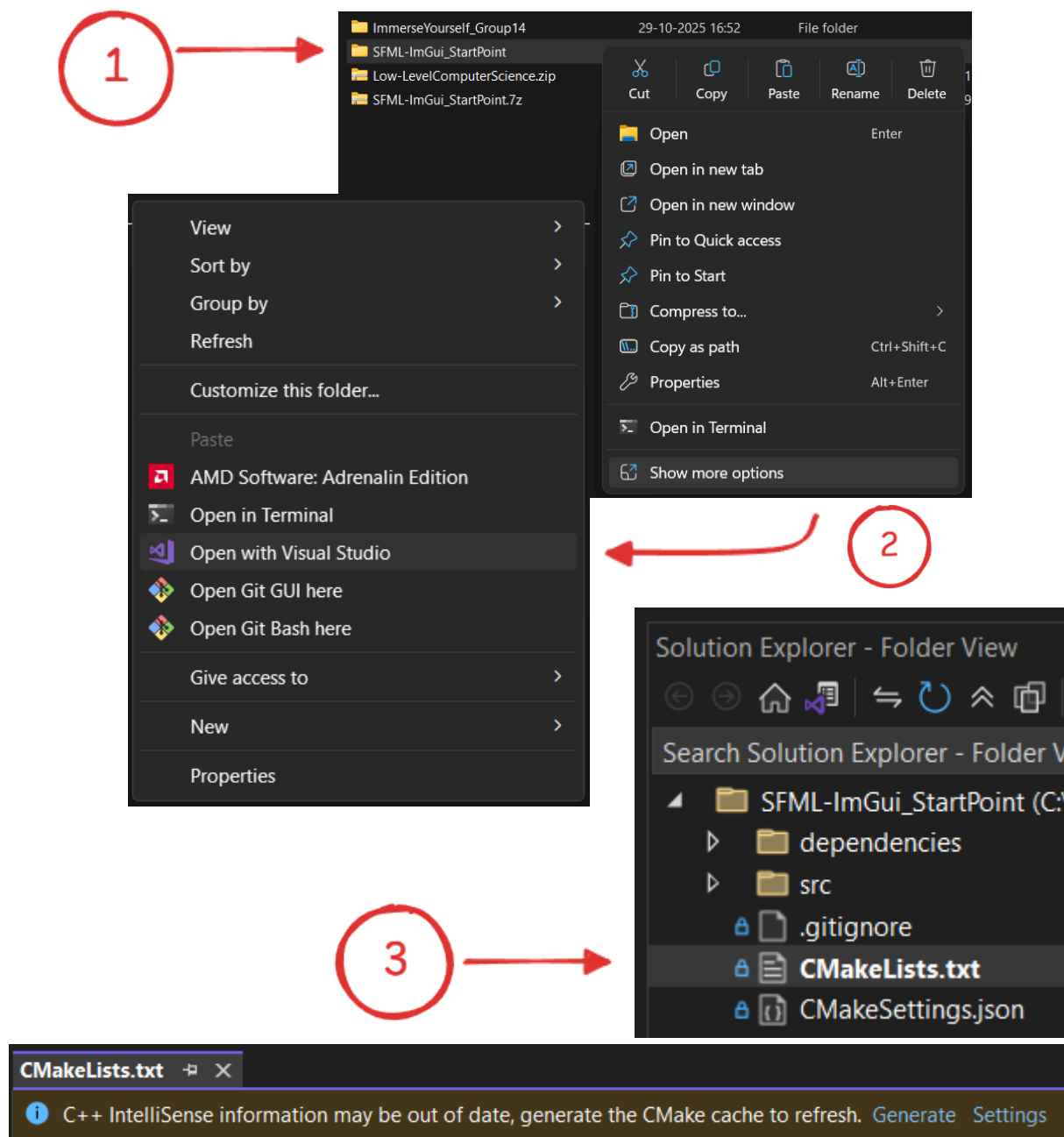
Dit heeft naderhand wel wat tijd geëist vanwege het steeds ombouwen van de gegeven Start Point bestanden per opdracht. Gezien ik een enkele Main loop heb en dus moest denken over folder structuur, main extensie en flow daarvan. Maar daardoor heb ik wel veel geleerd over het managen van meerdere bestanden in een enkel project en kon ik al gelijk vertrouwen kweken van elke gegeven Start Point code.



Project Set-Up (#Handleiding 1/2)

Mijn Handleiding

Dit is mijn manier van het project gebruiken / builden & runnen. Ik heb gedurende het project verschillende manieren via CMake geprobeerd: standalone versie, console, via visual studio zelf. Maar ik merkte voor mezelf dat dit voor mij de makkelijkste manier was, natuurlijk kan je je eigen methode toepassen, heb het allemaal vrij basic gehouden en uiteindelijk doet CMake toch de heavy lifting.



Project Set-Up (#Handleiding 2/2)

5

6

7

8

9

10

SET-UP KLAAR!

The screenshots show the following steps:

- Clicking the play button next to `low-level_setup.exe (src\Debug\low-level_setup.exe)` in the Solution Explorer.
- Clicking the `Show/Hide Debug Targets...` button.
- In the `Show or Hide Debug Targets` dialog, clicking the `Select All` checkbox.
- In the same dialog, ensuring `low-level_setup.exe (src\Debug\low-level_setup.exe)` is checked.
- Clicking the `OK` button.
- Clicking the play button next to `low-level_setup.exe (src\Debug\low-level_setup.exe)` in the Solution Explorer.

Libzmq Missing (#Oplossing)

Uitleg

Ik had problemen met de libzmq library die ontbrak na het bouwen & runnen van dit project, deze oplossing was een paar dagen voor de gene van de docenten ontdekt. Hij is nogal ruw, maar functioneel en zelf gevonden, dus heb hem aangehouden. In het kort is het enigste wat je doet, is handmatig de .dll bestand kopiëren en in de de folder met de .exe file plakken.

Stappenplan:

0. Probleem met missende "libzmq-v143-mt-gd-4_3_5.dll":
 - > Maak een **volledige build** eerst..
 - > Probeer hem daarna te **runnen**..
 - > Als je een pop-up hebt gekregen, creëert die de bin folder met de .dll bestand.
 1. Ga naar:
 - > ... \SFML-ImGui_StartPoint\out\build\x64-Debug_deps\libzmq-build\bin\Debug
 - > Vind en **kopieer**: "libzmq-v143-mt-gd-4_3_5.dll" bestand. (**Ctrl + C**)
 2. Ga naar:
 - > ... \SFML-ImGui_StartPoint\out\build\x64-Debug\src\Debug
 - > **Plak** de "libzmq-v143-mt-gd-4_3_5.dll" kopie in deze folder. (**Ctrl + V**)
 3. Ta-Da, alles zou nu moeten werken zoals het hoort!
-

1. Game of Life (#Toelichting)

Aanpak

In mijn Game of Life opdracht heb ik de Start Point overgenomen en zo veel mogelijk gemodulariseerd. Divide & Conquer methodiek, daarna alle onderdelen na gelopen en ietsjes verbeterd. Voor een goede framerate raad ik 250x250 aan, je kan hoger maar dan dalen de frames drastisch.

Modulariteit

Ik vond het interessant dat je regels modulair kon toepassen in de start point bestand en daar dan zulke verschillende resultaten uit kon krijgen. Ik heb een absolute verslaving aan modulaire systemen, dus dit greep me wel en heb vervolgens hier mijn eigen uitbreiding op uitgevoerd. Door ook de Neighbour Cell Offsets modulair te maken, zodat je kan bepalen welke Neighbours een Cell checkt, waardoor je nog meer resultaten krijgt. Mijn favoriet is bij ver: de Ninja Star Offsets, die geeft in bepaalde combinaties hele gekke resultaten.

```

class NinjaStarOffsets : public CellOffsets {
public:
    std::vector<std::pair<int, int>> getOffsets() override
    {
        static const std::vector<std::pair<int, int>> offsets =
        {
            {1,-2},
            {-2,-1}, {0,-1}, {1,0},
            {-1,0}, {0,1}, {2,1},
            {-1,2},
        };
        return offsets;
    }
};

```

```

static const std::vector<std::pair<int, int>> offsets =
{
    {-2,-2}, {-1,-2}, {0,-2}, {1,-2}, {2,-2},
    {-2,-1}, {-1,-1}, {0,-1}, {1,-1}, {2,-1},
    {-2,0}, {-1,0}, {1,0}, {2,0},
    {-2,1}, {-1,1}, {0,1}, {1,1}, {2,1},
    {-2,2}, {-1,2}, {0,2}, {1,2}, {2,2}
};

```

```

static const std::vector<std::pair<int, int>> offsets =
{
    {-2,-2}, {-1,-2}, {0,-2}, {1,-2}, {2,-2},
    {-2,-1}, {2,-1},
    {-2,0}, {2,0},
    {-2,1}, {2,1},
    {-2,2}, {-1,2}, {0,2}, {1,2}, {2,2}
};

```

1. Game of Life (#Set-Up)

Voor Playtesting

In de Game Of Life Assignment.h kan je currentWorld aanpassen met je eigen Inputs, die allemaal andere Outputs geven, heb er al drie voorbeelden staan. De eerste twee waarden zijn voor het speelveld, verticale grootte en horizontale grootte. De derde is de grootte voor het renderen van de cellen, minimaal 1 voor 1 pixel. De vierde waarde is voor het percentage levende cellen in het begin. De vierde modulaire regels van de game of life en vijfde mijn modulaire offsets voor elke cell.

```
//WorldData currentWorld = WorldData(250, 250, 2, 75, new ConwayRules, new MooreOffsets);
//WorldData currentWorld = WorldData(250, 250, 1, 60, new HighLifeRules, new SpacedStarOffsets);
WorldData currentWorld = WorldData(250, 250, 2, 80, new DayNight, new NinjaStarOffsets);
```

4_GameOfLife
HeaderFiles
Offsets
LargeSquareOffsets.h
MooreOffsets.h
NinjaStarOffsets.h
RoundedLargeSquareOffsets.h
SpacedSquareOffsets.h
SpacedStarOffsets.h
VonNeumannOffsets.h
Rules
ConwayRules.h
DayNight.h
HighLifeRules.h
PedestrianLife.h
CellData.h
CellOffsets.h
CellRules.h
FPSCounter.h
GameOfLifeAssignment.h
WorldData.h
WorldGenerator.h
SourceFiles
GameOfLifeAssignment.cpp
WorldGenerator.cpp

```
class CellOffsets
{
public:
    virtual ~CellOffsets() = default;
    CellOffsets() = default;

    virtual std::vector<std::pair<int, int>> getOffsets() = 0;
};
```

```
// Way Faster.. Like 60+ fps!
std::for_each(
    (cells.begin(),
     cells.end(),
     [&](CellData& cell)
    {
        int cellIndex = &cell - &cells[0];
        int x = cellIndex % width;
        int y = cellIndex / width;

        sf::Color color = currentWorld.GetCurrentCellAt(x, y).GetAliveStatus() ? sf::Color::Cyan : sf::Color::Blue;

        float left = float(x * size);
        float top = float(y * size);
        float right = float((x + 1) * size);
        float bottom = float((y + 1) * size);

        int vertexIndex = cellIndex * 6;

        // First triangle
        vertices[vertexIndex + 0].position = sf::Vector2f(left, top);
        vertices[vertexIndex + 1].position = sf::Vector2f(right, top);
        vertices[vertexIndex + 2].position = sf::Vector2f(right, bottom);

        // Second triangle
        vertices[vertexIndex + 3].position = sf::Vector2f(left, top);
        vertices[vertexIndex + 4].position = sf::Vector2f(right, bottom);
        vertices[vertexIndex + 5].position = sf::Vector2f(left, bottom);

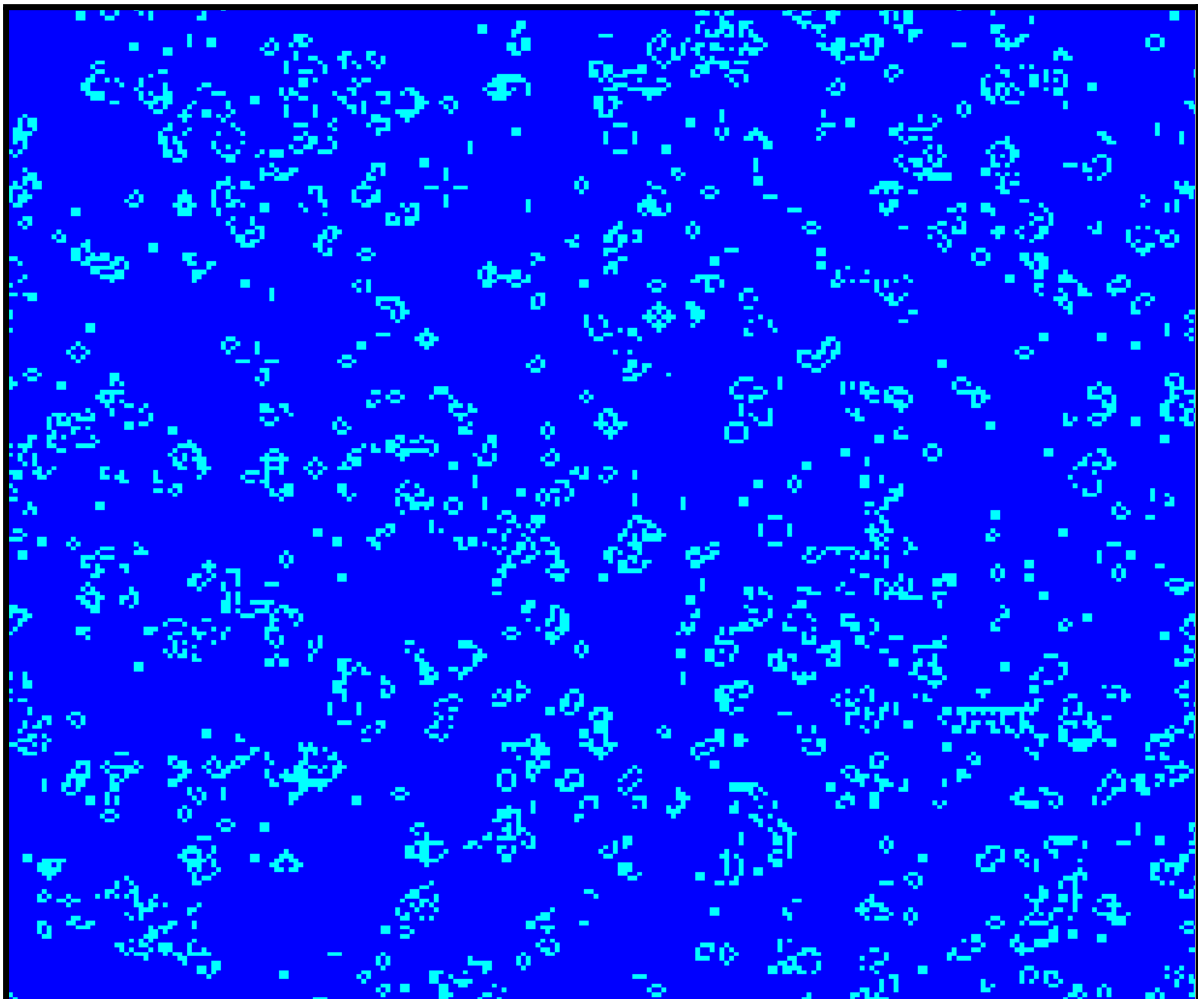
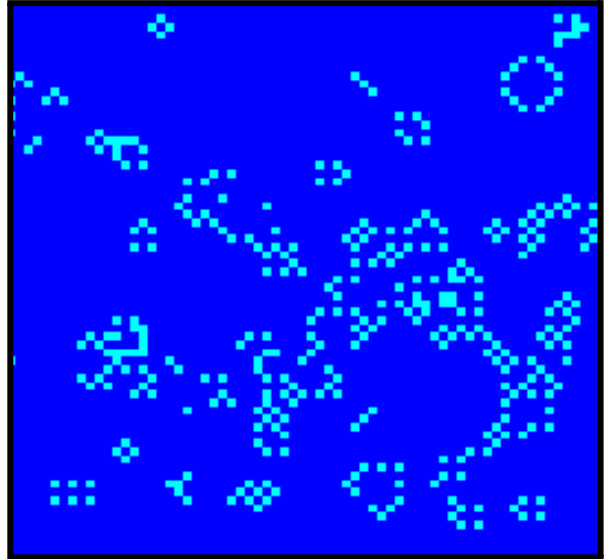
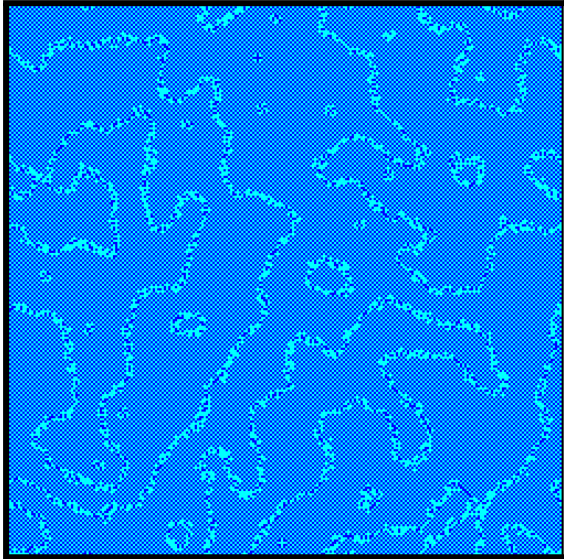
        for (int i = 0; i < 6; ++i) vertices[vertexIndex + i].color = color;
    });
```

```
std::for_each(
    (current.begin(),
     current.end(),
     [&](const CellData& cell)
    {
        int index = &cell - &current[0];
        int x = index % w;
        int y = index / w;

        CellData& bufferCell = buffer[index];
        int aliveNeighborsCount = CountAliveNeighborsAt(worldData, x, y, w, h);
        bufferCell.SetAliveNeighborCount(aliveNeighborsCount);

        bool survivedRules = worldData.GetCellRules()->getNewState(cell.GetAliveStatus(), aliveNeighborsCount);
        bufferCell.SetAliveStatus(survivedRules);
    });
```

1. Game of Life (#Highlights)



2. Chat Client & Server (#Toelichting)

Aanpak

In mijn Chat Client (& Server) opdracht heb ik de Start Points overgenomen en zo identiek mogelijk gehouden, gezien ik in het begin heel veel moeite had om alleen al de libraries werkend te krijgen. Daarna heb ik alle minimale eisen erin verwerkt en eigenlijk niet heel veel extra's.

Aparte Server Thread

heb ik ook later nog de server in zijn geheel op een aparte thread geplaatst gezien ik anders niet met mijn ImGui paneel interacties kon maken, zoals "Return" om een andere opdracht te openen.

Eigen Toevoeging

Voor mijn eigen toevoeging ben ik voor iets simpels gegaan, een wachtwoord (Passkey123") om toegang te krijgen tot de server. De Client vult het wachtwoord samen met de juiste server in beide textboxes. Dan tijdens het maken van de connectie naar de server check hij deze of het klopt en op basis daarvan geeft de server toegang of weigert hij de Client.

```
if (!connected)
{
    ImGui::InputText("Server", server_buffer, sizeof(server_buffer));
    ImGui::InputText("Password", password_buffer, sizeof(password_buffer));

    if (ImGui::Button("Connect"))
        connect_to_server();
}
```

```
void ChatServer::handle_password(const std::string& identity, const std::string& key)
{
    // Password Check.
    if (key == "Passkey123")
    {
        // Accepted Entry.
        std::cout << "Login Server Accepted." << std::endl;
        handle_connect(identity);
        send_message(identity, "LOGIN_ACCEPTED");
    }
    else
    {
        // Denied Entry.
        std::cout << "Login Server Denied." << std::endl;
        send_message(identity, "LOGIN_DENIED");
    }
}
```

2. Chat Client & Server (#Set-Up)

Voor Playtesting

Om zowel de Chat Client & Server te testen kan je simpelweg vanaf Visual Studio meerdere vensters openen via de knop "Start Without Debugging" (of Ctrl + F5). Dan navigeer je naar de "Chat Server Assignment" en die laat je runnen op een van de tabs. De rest kan je dan de "Chat Client Assignment" openen en daar kan je dan connectie en al het andere met de server maken. Voor het sturen van Private Messages naar een ander type je simpelweg hun naam in de target box, dan gaat het tijdens het versturen naar die persoon.

The image displays a Visual Studio code editor with several code snippets and a file explorer view.

Code Snippets:

```
// Runs until server->running == false.
serverThread = std::thread([this]()
{
    server->run();
});

// Signal the server to stop.
if (server)
    server->running = false;

if (msg_type == "CONNECTED") { ... }
else if (msg_type == "DISCONNECTED") { ... }
else if (msg_type == "PING") { ... }
else if (msg_type == "USERNAME_SET") { ... }
else if (msg_type == "USERNAME_TAKEN") { ... }
else if (msg_type == "PRIVATE_MSG") { ... }
else if (msg_type == "PUBLIC_MSG") { ... }
else if (msg_type == "MSG_DELIVERED") { ... }
else if (msg_type == "USER_NOT_FOUND") { ... }
else if (msg_type == "NOT_AUTHENTICATED") { ... }
else if (msg_type == "PUBLIC_MSG" && msg.size() >= 3) { ... }
else if (msg_type == "JOINED" && msg.size() >= 2) { ... }
else if (msg_type == "LEFT" && msg.size() >= 2) { ... }
else if (msg_type == "LOGIN_ACCEPTED") { ... }
else if (msg_type == "LOGIN_DENIED") { ... }

// Debug flags for development.
ImGui::Text
(
    // Suggestion from ChatGPT, absolutely love it!
    "Debug: connected=%s authenticated=%s identity=%s name=%s",
    connected ? "true" : "false",
    authenticated ? "true" : "false",
    identity.c_str(),
    username.c_str()
);
```

File Explorer View:

- 1_ChatClient
 - HeaderFiles
 - ChatClient.h
 - ChatClientAssignment.h
 - SourceFiles
 - ChatClient.cpp
 - ChatClientAssignment.cpp
- 1_ChatServer
 - HeaderFiles
 - ChatServer.h
 - ChatServerAssignment.h
 - SourceFiles
 - ChatServer.cpp
 - ChatServerAssignment.cpp

2. Chat Client & Server (#Highlights)

▼ Chat Client

Debug: connected=false authenticated=false identity= name=
 Server
 Password

▼ Chat Client

Debug: connected=true authenticated=false identity=Client_8009 name=
Connected to: tcp://localhost:5555

Set your username:

 Username

▼ Chat Client

Debug: connected=true authenticated=true identity=Client_8009 name=Wessel
Connected to: tcp://localhost:5555

Logged in as: Wessel

ChatClient Identity: Client_8009
Connecting to tcp://localhost:5555...
[PRIVATE] Connected to server.
[PRIVATE] Login accepted, correct password.
[PRIVATE] You are now known as: Wessel
[PUBLIC] Wessel joined the chat.
[PUBLIC] Wessel: Hallo Allemaal!
[PRIVATE] Tester -> Wessel: Wie ben jij?
[PUBLIC] Tester: Ooowh vergeten, hallooo allemaal!
[PUBLIC] Wessel: Ik ben een god!
[PRIVATE] Wessel -> Tester: IK BEN WESSEL JONGEN!

Target
 Message

3. Collision Crisis (#Toelichting)

Aanpak

In mijn Collision Crisis opdracht heb ik de Start Points overgenomen en zo identiek mogelijk gehouden, daarna heb ik Spatial Hashing toegepast en uitgebreid getest, door resultaten daarvan te vergelijken, voor de rest heb ik het vrij minimaal gehouden.

Data

De data is niet super accuraat, kwam ik later pas achter toen ik bezig was met de Memory & Cache Optimizations opdracht. Had jammer genoeg geen tijd meer om nog correcties en opnieuw een aantal testjes te doen. Dus neem de tabel hieronder met een korreltje zout, vooral de tijd en waarschijnlijk ook de snelheid, rest zou wel moeten kloppen.

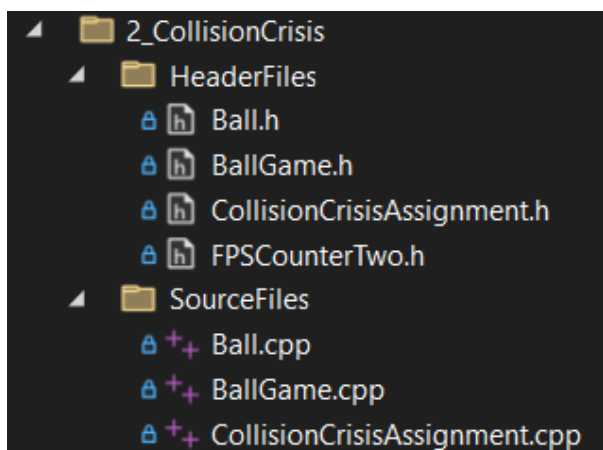
CELL SIZE	0	5	7,5	10	12,5	15	17,5
HASHES AVERAGE	0	2405	2257	2062	1817	1570	1340
FPS AVERAGE	19	39	38	39	38	39	39
SPEED AVERAGE	73.008ms	4.044ms	4.564ms	4.533ms	5.100ms	5.244ms	5.378ms
TIME AVERAGE	0.096s	0.027s	0.031s	0.028s	0.031s	0.031s	0.031s

3. Collision Crisis (#Set-Up)

Voor Playtesting

Je kan in de BallGame.h de Cell Size aanpassen om andere Spatial Hashing resultaten te krijgen, alle performance data is zichtbaar in de console per generatie en na 1000 generaties stopt de simulatie en dan is er ook een gemiddelde per generatie in de console zichtbaar.

```
const float cellSize = 5.0f;
```



```
// Collision check using nearby cells.
for (size_t i = 0; i < balls.size(); ++i)
{
    auto pos1 = balls[i].shape.getPosition();
    int cellX = static_cast<int>(pos1.x / cellSize);
    int cellY = static_cast<int>(pos1.y / cellSize);

    for (int ox = -1; ox <= 1; ++ox)
    {
        for (int oy = -1; oy <= 1; ++oy)
        {
            auto h = hash(cellX + ox, cellY + oy);
            if (!hashGrid.count(h)) continue;

            for (size_t j : hashGrid[h]) { ... }
        }
    }
}
```

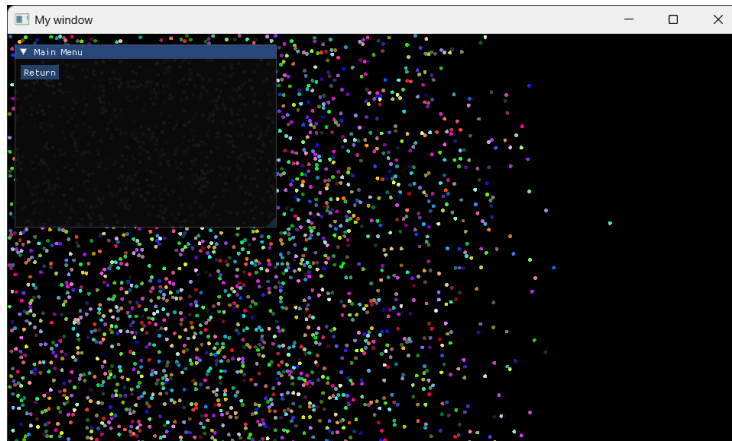
```
// Assign each ball to a spatial hash cell based on its position.
for (size_t i = 0; i < balls.size(); ++i)
{
    auto pos = balls[i].shape.getPosition();
    int cellX = static_cast<int>(pos.x / cellSize);
    int cellY = static_cast<int>(pos.y / cellSize);

    // Insert this index into the corresponding cell in the hash grid.
    hashGrid[hash(cellX, cellY)].push_back(i);
}
```

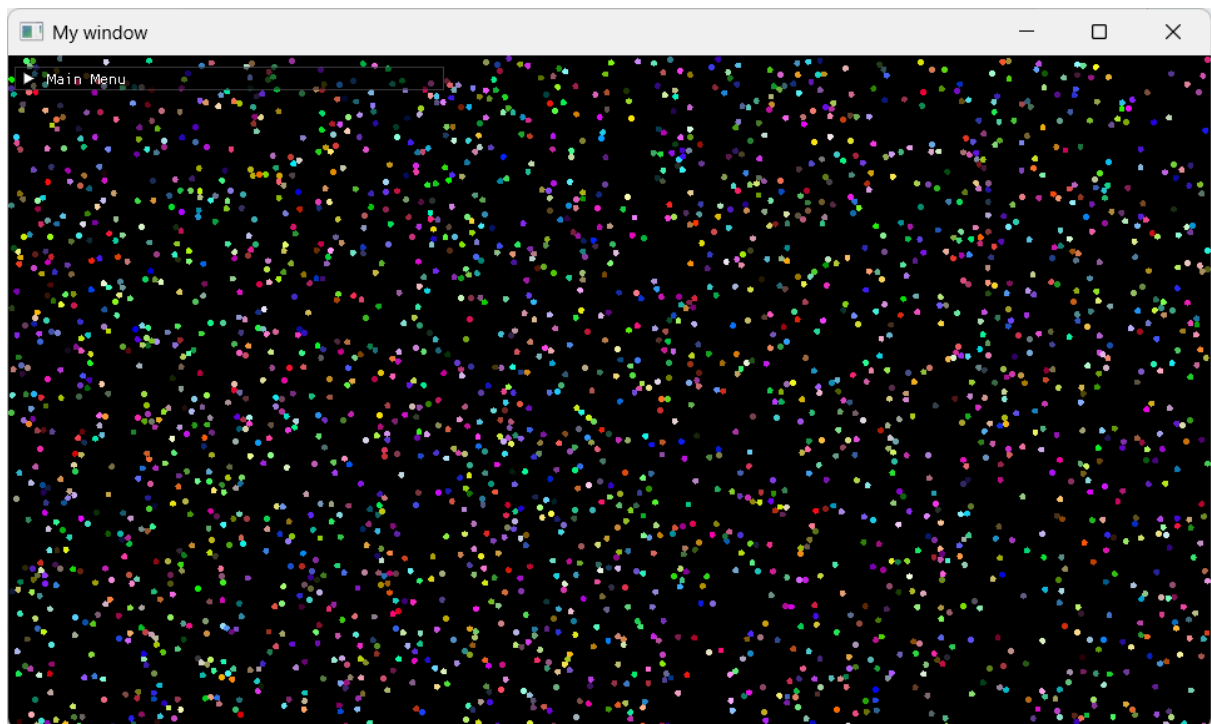
```
void BallToBallOriginal();
void BallToBallSpatialHashing();
```

```
std::vector<Ball> balls;
std::unordered_map<long long,
    std::vector<size_t>> hashGrid;
```

3. Collision Crisis (#Highlights)



```
[AVERAGE OF GENERATIONS]: 1000
[BALLS AVERAGE]: 2500
[SPEED AVERAGE]: 3.901ms
[TIME AVERAGE]: 0.027s
[FPS AVERAGE]: 39fps
[HASHES AVERAGE]: 2404
[CELLSIZE]: 5.000
```



```
[GEN]: 1
[BALLS]: 2500
[SPEED]: 3.456ms
[TIME]: 0.024s
[FPS]: 10fps
[HASHES]: 2236
[CELLSIZE]: 5.000
```

```
[GEN]: 666
[BALLS]: 2500
[SPEED]: 4.858ms
[TIME]: 0.040s
[FPS]: 30fps
[HASHES]: 2374
[CELLSIZE]: 5.000
```

```
[GEN]: 1000
[BALLS]: 2500
[SPEED]: 4.971ms
[TIME]: 0.039s
[FPS]: 31fps
[HASHES]: 2361
[CELLSIZE]: 5.000
```

4. Memory & Cache Optimization (#Toelichting)

Aanpak

In mijn Memory & Cache Optimization opdracht heb ik de Start Points overgenomen, vervolgens al het benodigde toegevoegd op basis van de minimale eisen. Daarna wist ik al dat ik ook de Multi-Threading opdracht hierin wilde maken en ben ik al vrij vroeg begonnen met de boel op te splitsen. Om uiteindelijk twee varianten te krijgen: eentje, die via AoS werkt en dan de ander via SoA.

~~Particle & GameObject~~ → Struct & Shared Physics

Gedurende het werkproces heb ik gekozen om de gegeven Particle- en GameObject code te verwijderen en een Particle Struct te gebruiken voor AoS. Vervolgens heb ik gezorgd dat beide varianten dezelfde Physics Functie gebruiken, zodat ze exact hetzelfde doen op dat gebied.

Fixed Seed & Fixed Time

Na een aantal testen kwam ik erachter dat als ik de Start Particles omhoog gooide dat de simulatie sneller klaar was, uiteindelijk een gekke timing artifact. Maar hierdoor heb ik wel geleerd hoe belangrijk het is om zoveel mogelijk dingen hetzelfde te houden als je iets specifiek aan het testen bent, zoals snelheid of memory tussen SoA en AoS. Dus hierna heb ik zowel de RNG als DeltaTime optioneel gemaakt en vanaf dat punt uitgezet. Opnieuw alles getest en met veel betere resultaten die een stuk minder schrikbarend waren.

Swap & Pop

Ik ben ook geswitched naar een Swap & Pop methode in plaats van Erase, die is iets sneller en voor mij gevoel leesbaarder. Ik had hem in het begin al veranderd en de testen van toen waren niet super accuraat, dus in hoeverre dit nog steeds zo is geen idee, waarschijnlijk alsnog sneller?

Memory Tracker

Deze was best lastig te creëren, was super complex en ver boven mijn niveau. Uiteindelijk met behulp van AI werkend gekregen door operators te gebruiken, wilde perse iets hebben dat via de console zou werken, gezien ik dan gemakkelijk de data kan oogsten. En ik hoor hem al aankomen.. Nee, AI heeft niet de code gegenereerd, wel snippets als voorbeelden. Maar heb puur gevraagd om hulp en uitleg over bepaalde dingen / lastige problemen, een soort virtuele docent die je de goede kant op wijst en tegelijker feedback geeft op wat je laat zien, om zo jezelf te verbeteren.

4. Memory & Cache Optimization (#Set-Up)

Voor Playtesting

Je kan in MemoryAndCacheOptiAssignment.h een aantal variabelen aanpassen om andere resultaten te krijgen, raad aan om alleen het gedeelte met "Important variables" aan te passen.

```
// Important variables
sf::Vector2f origin = { 480.0f, 270.0f };
int startParticleCount = 1000;
WayOfStorage thisWayOfStorage = WayOfStorage::AoS;
bool thisMultiThreading = true;
bool thisRandomSeed = false;
bool thisFixedTime = true;
float fixedTimeStep = 0.1f;
```

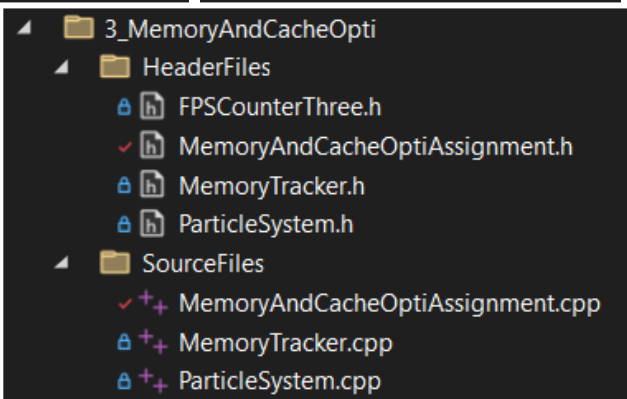
```
> struct RandomRanges { ... };
> struct SimpleParticle { ... };
```

```
// Remove dead particles using swap & pop.
for (size_t i = 0; i < particles.size(); )
{
    if (particles[i].lifetime <= 0)
    {
        std::swap(particles[i], particles.back());
        particles.pop_back();
    }
    else ++i;
}
```

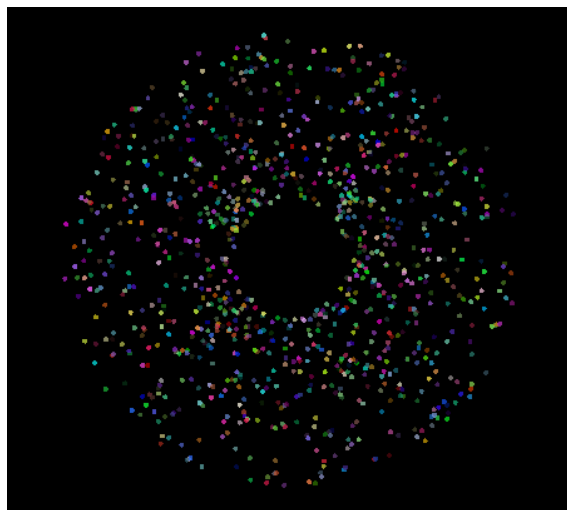
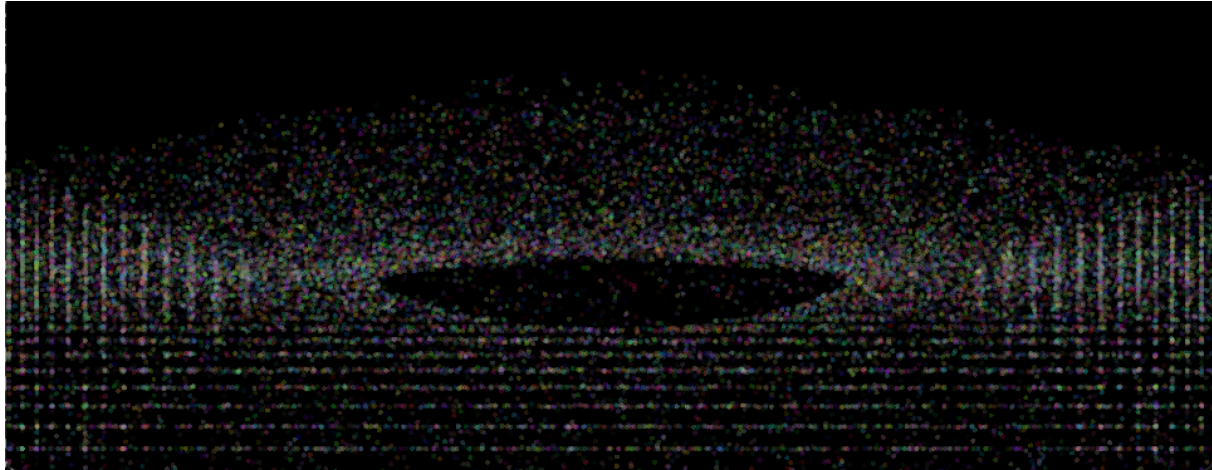
```
enum class WayOfStorage
{
    AoS,
    SoA
};
```

```
void ParticleSystem::UpdateSharedPhysics
(
    sf::Vector2f& position,
    sf::Vector2f& velocity,
    float& lifetime,
    float maxLifetime,
    sf::Color& color,
    float deltaTime
) { ... }
```

```
// Randomness:
RandomRanges randomRanges;
unsigned fixedSeed = 12345;
std::mt19937 rng;
```



4. Memory & Cache Optimization (#Highlights)



```
[GENERATIONS]:      52
[SIM SPEED]:       71.469ms
[SIM TIME]:        5.200s
[AVERAGE FPS]:    10.038fps
[START PARTICLES]: 100000
[TIMING]:          FIXED
[WAY OF STORAGE]:  AoS
[MEMORY AVERAGE]: 2777.33 KB
[MEMORY CURRENT]:  2777.81 KB
[MEMORY PEAK]:     2780.46 KB
[ALLOC COUNT]:     50468663
[DEALLOC COUNT]:   50467886
[THREADS USED]:    12
```

```
[GEN]:             21
[SPEED]:           1.877ms
[TIME]:            2.100s
[FPS]:             10.000fps
[PARTICLES]:       74954
[TIMING]:          FIXED
[WAY OF STORAGE]:  AoS
[MEMORY CURRENT]:  2776.86 KB      +0.00 KB
[MEMORY PEAK]:     2779.46 KB
[ALLOC COUNT]:     15909592      +619734
[DEALLOC COUNT]:   15908824      +619734
[THREADS USED]:    12
```

5. Multi-Threading (#Toelichting)

Aanpak

In mijn Memory & Cache Optimization opdracht heb ik twee extra functies aangemaakt voor AoS & SoA met multi-threading. Waardoor ik op precies 4 verschillende mogelijkheden kwam, SoA of Soa en single-threading of multi-threading. Vervolgens ben ik gaan kijken waar ik dit het beste kon toepassen, ik wilde het kleinschalig houden dus heb ik gekozen om de Physics Update workload te verspreiden. Voor het multithreading systeem heb ik de "mtrebi" ThreadPool gebruikt.

Chunks & Timing

In plaats van alles los in een queue te stoppen, heb ik gekozen om alles op te delen in chunks op basis de hoeveelheid particles en dat dan gelijk te verspreiden over al mijn 12 threads. Gezien al de data ongeordend is er vrijwel geen synchronisatie / locking nodig, daarnaast zijn ze toch allemaal alleen in hun eigen chunk bezig. Pas als alle threads klaar zijn met hun gegeven chunk, gaat de code verder als een enkele thread.

SoA & AoS

Ik had verwacht dat bij beide een unieke implementatie nodig was, maar dat was niet zo. Voor mij de eerste keer dat ik Multi-Threading heb geïmplementeerd en het is verbaasd wekkend modulair. Jammer dat ik niet meer tijd had om mezelf er nog meer in te verdiepen!

ThreadPool Instance

Een hele domme fout in het begin maakte ik de "mtrebi" ThreadPool altijd in de Particle System aan, maar dit zorgde na het opnieuw opstarten voor een hele onduidelijke error. Waar ik een behoorlijk aantal uur mee bezig ben geweest uiteindelijk kwam ik erachter dat het kwam omdat ik de ThreadPool kopieerde. Blijkbaar moet je deze echt maar een keer aanmaken, dus heb ik er toen een Static Instance van gemaakt die aan het begin van de Main plaatsvindt. Hierna werkte alles perfect en was het smooth sailing.

Dingen die mij opvielen op basis van mijn tests

-> Op de volgende vier bladzijden staan de tests per variant gedocumenteerd..!

1. Voor kleine hoeveelheden is er weinig verschil tussen alle vier de varianten.
2. Single-threaded simulaties worden trager naarmate het aantal particles stijgt.
3. Multi-threaded simulaties geven bij grotere workloads een flinke snelheidswinst.
4. In grotere simulaties met fixed timing is er een extra generatie, unieke artifact.
5. SoA is efficiënter in (de)allocaties en dus cache, maar is ook betrouwbaarder.
6. AoS heeft soms apart hoge (de)allocaties in vergelijking met de rest.
7. SoA gebruikt iets meer geheugen per particle.

SoA & Single-Threaded (#A-Tests)

[SEED]:	FIXED	← = Preciezer dan RNG..
[TIMING]:	FIXED	← = Preciezer dan DeltaTime..
[WAY OF STORAGE]:	SoA	
[THREADS USED]:	1	
[GENERATIONS]:	51	
[SIM SPEED]:	2.998ms	→ = Baseline..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	47.020fps	→ = Baseline..
[START PARTICLES]:	1000	← == 0x Particles..
[MEMORY AVERAGE]:	63.40 KB	→ = Baseline..
[MEMORY CURRENT]:	63.40 KB	→ = Baseline..
[MEMORY PEAK]:	66.18 KB	→ = Baseline..
[ALLOC COUNT]:	276763	→ = Baseline..
[DEALLOC COUNT]:	276098	→ = Baseline..
[GENERATIONS]:	51	
[SIM SPEED]:	24.128ms	→ ≈ 8,1x meer dan base..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	20.373fps	→ ≈ 0,4x minder dan base..
[START PARTICLES]:	10000	← == 10x Particles..
[MEMORY AVERAGE]:	317.68 KB	→ ≈ 5,0x meer dan base..
[MEMORY CURRENT]:	317.68 KB	→ ≈ 5,0x meer dan base..
[MEMORY PEAK]:	349.34 KB	→ ≈ 5,3x meer dan base..
[ALLOC COUNT]:	2533049	→ ≈ 9,2x meer dan base..
[DEALLOC COUNT]:	2532384	→ ≈ 9,2x meer dan base..
[GENERATIONS]:	52	← += Extra generatie.. RNG?
[SIM SPEED]:	226.553ms	→ ≈ 75.6x meer dan base..
[SIM TIME]:	5.200s	← += Extra tijd.. RNG?
[AVERAGE FPS]:	10.077fps	→ ≈ 0.2x minder dan base..
[START PARTICLES]:	100000	← == 100x Particles..
[MEMORY AVERAGE]:	2921.07 KB	→ ≈ 46,1x meer dan base..
[MEMORY CURRENT]:	2921.07 KB	→ ≈ 46,1x meer dan base..
[MEMORY PEAK]:	3279.68 KB	→ ≈ 49,6x meer dan base..
[ALLOC COUNT]:	25212634	→ ≈ 91,0x meer dan base..
[DEALLOC COUNT]:	25211918	→ ≈ 91,3x meer dan base..

SoA & Multi-Threaded (#B-Tests)

[SEED]:	FIXED	← = Preciezer dan RNG..
[TIMING]:	FIXED	← = Preciezer dan DeltaTime..
[WAY OF STORAGE]:	SoA	
[THREADS USED]:	12	← = Laptop Max
[GENERATIONS]:	51	
[SIM SPEED]:	8.810ms	→ = Baseline..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	45.706fps	→ = Baseline..
[START PARTICLES]:	1000	← == 0x Particles..
[MEMORY AVERAGE]:	67.94 KB	→ = Baseline..
[MEMORY CURRENT]:	68.41 KB	→ = Baseline..
[MEMORY PEAK]:	71.40 KB	→ = Baseline..
[ALLOC COUNT]:	273844	→ = Baseline..
[DEALLOC COUNT]:	273144	→ = Baseline..
[GENERATIONS]:	51	
[SIM SPEED]:	14.836ms	→ ≈ 1,7x meer dan base..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	20.902fps	→ ≈ 0,5x minder dan base..
[START PARTICLES]:	10000	← == 10x Particles..
[MEMORY AVERAGE]:	322.10 KB	→ ≈ 4,7x meer dan base..
[MEMORY CURRENT]:	322.19 KB	→ ≈ 4,7x meer dan base..
[MEMORY PEAK]:	349.40 KB	→ ≈ 4,9x meer dan base..
[ALLOC COUNT]:	2540925	→ ≈ 9,3x meer dan base..
[DEALLOC COUNT]:	2540223	→ ≈ 9,3x meer dan base..
[GENERATIONS]:	52	← += Extra generatie.. RNG?
[SIM SPEED]:	87.189ms	→ ≈ 9.9x meer dan base..
[SIM TIME]:	5.200s	← += Extra tijd.. RNG?
[AVERAGE FPS]:	10.077fps	→ ≈ 0.2x minder dan base..
[START PARTICLES]:	100000	← == 100x Particles..
[MEMORY AVERAGE]:	2924.78 KB	→ ≈ 43.0x meer dan base..
[MEMORY CURRENT]:	2924.81 KB	→ ≈ 42.8x meer dan base..
[MEMORY PEAK]:	3279.70 KB	→ ≈ 45.9x meer dan base..
[ALLOC COUNT]:	25219448	→ ≈ 92.1x meer dan base..
[DEALLOC COUNT]:	25218725	→ ≈ 92.3x meer dan base..

AoS & Single-Threaded (#C-Tests)

[SEED]:	FIXED	← = Preciezer dan RNG..
[TIMING]:	FIXED	← = Preciezer dan DeltaTime..
[WAY OF STORAGE]:	AoS	
[THREADS USED]:	1	
[GENERATIONS]:	51	
[SIM SPEED]:	2.329ms	→ = Baseline..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	40.980fps	→ = Baseline..
[START PARTICLES]:	1000	← == 0x Particles..
[MEMORY AVERAGE]:	62.95 KB	→ = Baseline..
[MEMORY CURRENT]:	62.95 KB	→ = Baseline..
[MEMORY PEAK]:	63.90 KB	→ = Baseline..
[ALLOC COUNT]:	320363	→ = Baseline..
[DEALLOC COUNT]:	319704	→ = Baseline..
[GENERATIONS]:	51	
[SIM SPEED]:	19.851ms	→ ≈ 8.5x meer dan base..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	20.902fps	→ ≈ 0.5x minder dan base..
[START PARTICLES]:	10000	← == 10x Particles..
[MEMORY AVERAGE]:	309.21 KB	→ ≈ 4.9x meer dan base..
[MEMORY CURRENT]:	309.21 KB	→ ≈ 4.9x meer dan base..
[MEMORY PEAK]:	310.16 KB	→ ≈ 4.9x meer dan base..
[ALLOC COUNT]:	2535761	→ ≈ 7.9x meer dan base..
[DEALLOC COUNT]:	2535097	→ ≈ 7.9x meer dan base..
[GENERATIONS]:	52	← += Extra generatie.. RNG?
[SIM SPEED]:	189.276ms	→ ≈ 81.3x meer dan base..
[SIM TIME]:	5.200s	← += Extra tijd.. RNG?
[AVERAGE FPS]:	10.077fps	→ ≈ 0.3x minder dan base..
[START PARTICLES]:	100000	← == 100x Particles..
[MEMORY AVERAGE]:	2771.63 KB	→ ≈ 441,0x meer dan base..
[MEMORY CURRENT]:	2772.13 KB	→ ≈ 44,0x meer dan base..
[MEMORY PEAK]:	2773.08 KB	→ ≈ 43.4x meer dan base..
[ALLOC COUNT]:	25214131	→ ≈ 78.7x meer dan base..
[DEALLOC COUNT]:	25213410	→ ≈ 78.8x meer dan base..

AoS & Multi-Threaded (#D-Tests)

[SEED]:	FIXED	← = Preciezer dan RNG..
[TIMING]:	FIXED	← = Preciezer dan DeltaTime..
[WAY OF STORAGE]:	AoS	
[THREADS USED]:	12	← = Laptop Max..
[GENERATIONS]:	51	
[SIM SPEED]:	8.868ms	→ = Baseline..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	44.980fps	→ = Baseline..
[START PARTICLES]:	1000	← == 0x Particles..
[MEMORY AVERAGE]:	67.54 KB	→ = Baseline..
[MEMORY CURRENT]:	67.88 KB	→ = Baseline..
[MEMORY PEAK]:	70.47 KB	→ = Baseline..
[ALLOC COUNT]:	277316	→ = Baseline..
[DEALLOC COUNT]:	276623	→ = Baseline..
[GENERATIONS]:	51	
[SIM SPEED]:	12.973ms	→ ≈ 1.5x meer dan base..
[SIM TIME]:	5.100s	
[AVERAGE FPS]:	20.725fps	→ ≈ 0.5x minder dan base..
[START PARTICLES]:	10000	← == 10x Particles..
[MEMORY AVERAGE]:	314.20 KB	→ ≈ 4.7x meer dan base..
[MEMORY CURRENT]:	314.29 KB	→ ≈ 4.6x meer dan base..
[MEMORY PEAK]:	316.57 KB	→ ≈ 4.5x meer dan base..
[ALLOC COUNT]:	5161751	→ ≈ 19x meer dan base.. !?!
[DEALLOC COUNT]:	5161053	→ ≈ 19x meer dan base.. !?!
[GENERATIONS]:	52	← += Extra generatie.. RNG?
[SIM SPEED]:	68.937ms	→ ≈ 7.8x meer dan base..
[SIM TIME]:	5.200s	← += Extra tijd.. RNG?
[AVERAGE FPS]:	10.077fps	→ ≈ 0.2x minder dan base..
[START PARTICLES]:	100000	← == 100x Particles..
[MEMORY AVERAGE]:	2775.73 KB	→ ≈ 41.1x meer dan base..
[MEMORY CURRENT]:	2775.95 KB	→ ≈ 40.9x meer dan base..
[MEMORY PEAK]:	2779.17 KB	→ ≈ 39.5x meer dan base..
[ALLOC COUNT]:	25222027	→ ≈ 90.9x meer dan base..
[DEALLOC COUNT]:	25221282	→ ≈ 91.2x meer dan base..

Reflectie (#Persoonlijk)

Samengevat

Het was een heel druk vak, bomvol met informatie en ik merkte voor het eerst dat ik best veel moeite had om dingen (snel) te begrijpen en toe te passen. CMake leren was een heel proces en de libzmq library een absolute power drain om werkend te krijgen. Die twee dingen hebben voor mij de meeste tijd gekost, wat eigenlijk wel jammer is. Voor het mooie had ik ook nog een tweede iteratie over alles moeten doen en ook gelijk de code conventies hard moeten strak trekken, daar had ik nu geen tijd meer voor.

Groepsproject Factor

Daarnaast is dit een groepsproject blok en ik merk dat die dubbel de energie en tijd kosten in vergelijking met een solo project blok. Jammer genoeg liep het groepsproject ook weer niet soepel, nu als enige developer en al het werk dat bij mij terecht kwam. Ik heb geprobeerd zo min mogelijk mijn andere vakken hieronder te laten lijden (wat tweemaal vorig jaar wel was gebeurd), maar uiteindelijk heeft het toch weer zijn stempel gedrukt. Waardoor ik me niet verder kon verdiepen in de opdrachten en dus echt ben gaan min-maxen om het meeste met zo min mogelijk tijd te behalen, dit zorgt voor een hele stressvolle omgeving in de laatste paar weken van het blok. Dus gezien de omstandigheden had ik er denk ik ook niet meer uit kunnen halen dan dit resultaat, ondanks dat wel enorm veel geleerd en een gigantische sprong gemaakt in mijn ontwikkeling.

Algemene verbeterpunten, waar ik het volgende blok aan wil werken:

- Dieper vooraf analyseren op technologie i.p.v. er blind inspringen.
- Code Conventies handhaven vanaf het begin en goed nalopen.
- Code Conventies in het algemeen meer aandacht en tijd geven.
- Comments, comments en meer comments: ben te terughoudend geweest.
- Meer visualisaties toepassen: (Flow)charts / UML's / One-pagers.
- Tijdens mijn Development vak denken als Specialist in plaats van Generalist.
- Meer tijd besteden aan Development ontwikkeling in het algemeen.
- Feedback van docenten en studenten vragen en gebruiken. (I.p.v. → AI)
- Gedurende projecten bijlagen beter opslaan.
- + / -