Verslag

Project Optimalisatie

Naam en Studentnummer	Bryan Sok (4877144)
Periode	2
Versie	3.0
Inleverdatum	16-01-2024
Opmerking	Ik hoef alleen Concurrent Programmeren te herkansen.

Inleiding

Deze periode kregen we een programma dat heel slecht was geoptimaliseerd. Het programma was geschreven in C++. Het is een simulatie van 2 groepen tanks die elkaar aanvielen.

Helaas kon ik door persoonlijke omstandigheden de 2de periode vorig jaar niet meer afronden. OOP in C++ en Algoritmiek is me nog wel gelukt, met respectievelijk een 7,2 en 7,0, maar aan Concurrent Programmeren ben ik helaas niet meer toegekomen. Bij deze doe ik dus alsnog een poging.

Ondanks dat ik Algoritmiek en OOP wel heb afgerond, heb ik nog wel de Big-O notaties en de algoritmen in dit verslag gelaten. Ze zijn dus exact hetzelfde als in de vorige versie, maar horen natuurlijk wel bij de speedup van de simulatie. Dan krijgt u misschien een iets beter idee hoe zinvol de concurrency in dit geval was.

Alvast bedankt voor het nakijken!

Inhoudsopgave

1. Big O notaties

1.1 Time Complexity

Terrain.cpp

28 - 73	O(NM)	nested for-loop with different inputs
76 - 88	O(n^2)	nested for-loop
99 - 128	O(n^2)	nested for-loop
96 - 128	O(n^2)	Nested-for-loop
150 - 173	O(NM)	nested for-loop with different inputs
176 - 179	O(N)	Single for-loop, thus Linear
181 - 197	O(N)	Single for-loop, thus Linear
205 - 226	O(1)	Constant
231 - 241	O(1)	Constant

Particle_beam.cpp

13 -14	O(1)	Constant
22 - 25	O(1)	Constant
35	O(log n)	Division by 10

Explosion.cpp

6	O(1)	Constant
11	O(1)	Constant
16	O(log n)	Division by 2

Game.cpp

57-63	O(1)	Constant
66-70	O(n)	Single Loop
72-76	O(n)	Single Loop
98-109	O(n)	Single Loop
111	O(1)	Constant
117	O(1)	Constant
131-137	O(n)	Single Loop
140-160	O(n^2)	Nested Loop
163 - 180	O(n)	Single Loop
183 - 186	O(n)	Single Loop

193 - 200	O(n)	Single Loop
203 - 212	O(n)	Single Loop
215- 239	O(n^2)	Nested Loop

Rocket.cpp

16-20	O(1)	Constant
29-45	O(n^2)	Nested for lus

Tank.cpp

43-46	O(N)	Single for lus
48-50	O(1)	Constant
52-56	O(N)	Single for lus
60	O(1)	If zonder loop
62-71	O(N^2)	Nested lus
87-92	O(1)	Constant
94-97	O(1)	Constant
121-124	O(1)	Constant

Game.cpp

246-262	O(N^2)	Nested lus
264-279	O(NM)	Nested lus met meerdere inputs
291-302	O(NM)	Nested lus met meerdere inputs
387-408	O(NM)	Nested lus met meerdere inputs
401-404	O(N)	Single for lus
418-425	O(N)	Single for lus
429-435	O(N)	Single for lus
437-438	O(N)	Single for lus
447-460	O(N^2)	Nested lus
462-470	O(N)	Single for lus
476-482	O(N)	Single for lus

1.2 Space Complexity

Terrain.cpp

10 - 26	O(1)	Images declarations
29- 71	O(1)	Switch cases, no vectors, and a few declarations
102 - 137	O(1)	Switch case statement, no vectors
143 - 147	O(1)	4 Constant declarations, Constant
150- 192	O(n)	Contains a vector, so space depends on vector_size
194 - 213	O(n)	Contains a vector, so space depends on vector_size
216 - 242	O(1)	Returns an int, constant
245 - 257	O(1)	Has 2 int declarations, thus constant

Particle_beam.cpp

13 -17	O(1)	Sum of min and max, thus constant
22 - 25	O(n)	Has a vector, space depends on array
28 - 37	O(1)	Has 2 int declarations, thus constant

Explosion.cpp

4 - 7	O(1)	Returns something, constant
9 - 12	O(1)	Just an increment, thus constant
14 - 18	O(1)	No need for new memory, so constant

Game.cpp

0 - 42	O(1)	Single declarations
51 - 63	O(n)	Tanks Vector is used, memory depends on tanks
66-80	O(nm)	2 Vectors (Blue and Red) Used
95-111	O(n)	Usage of the Tanks Vector
115-119	O(1)	Single Return-statement
131-137	O(n)	Usage of tanks vector, memory usage dependent on vector
140-160	O(n)	Usage of tanks vector, memory usage dependent on vector
163 - 180	O(n)	Usage of tanks vector, memory usage dependent on vector
183 - 186	O(n)	Usage of smokes vector, memory usage dependent on vector
193 - 200	O(n)	Usage of tanks vector, memory usage dependent on vector
203 - 212	O(n)	Usage of tanks vector, memory usage dependent on vector
215- 239	O(n)	Usage of tanks vector, memory usage dependent on vector

Rocket.cpp

5-10	O(1)	Vectors
12-14	O(1)	Rocket
22-27	O(1)	Rocket draw

Tank.cpp

5-33	O(1)	Sprites
35-37	O(1)	Tank
58	O(1)	Vec2
73-85	O(N)	Route
99-111	O(1)	Removes health bar
113-119	O(N)	Draw the sprite with the facing based on this tanks movement direction
126-130	O(1)	Add some force in a given direction

Game.cpp

241-244	O(1)	Update rockets
283-284	O(1)	Remove exploded rockets with remove erase idiom
285-289	O(N)	Update particle beams
304-308	O(N)	Update explosion sprites and remove when done with remove erase idiom
310	O(1)	Explosion
319-320	O(1)	Clear the graphics window
322-323	O(1)	Draw screen

325-352	O(N)	Draws each rocket, smoke, partivle_beam
		and explosion in their corresponding list
354-362	O(N)	Draw forcefield (mostly for debugging, its kinda ugly..)
364-376	O(N)	Draw sorted health bars
381-385	O(1)	Insertion sort tanks health
413-416	O(1)	Draw health bars
493-495	O(1)	Print frame count

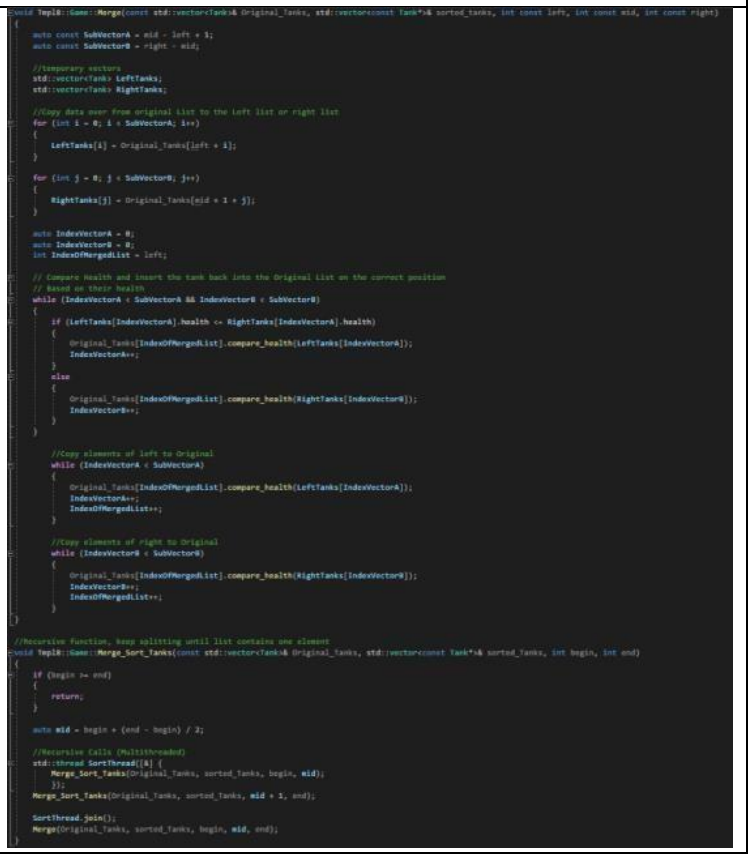
2. Algoritmiek

Toen ik het programma opstartte voor de 1ste keer, duurde het 12 minuten en 36 seconden voor het programma volledig klaar was.

2.1 Gebruikt algoritme 1

Naam algoritme	Dijkstra
Hoe gevonden?	Algoritmiek les 4
Hoeveel sneller?	1 minuut en 28 seconden
Screenshot code	

2.2 Gebruikt algoritme 2

Naam algoritme	Merge sort
Hoeveel sneller?	1 minuut en 38 seconden
Screenshot code	

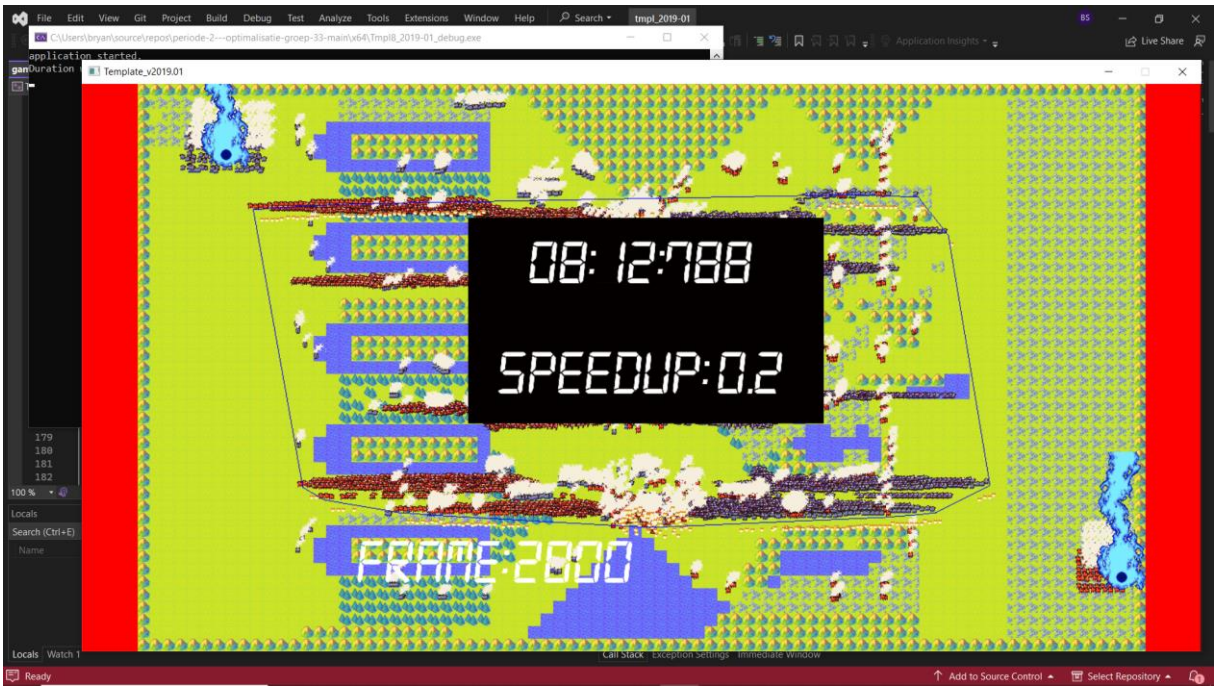
```
void Tmp10::Game::Merge(const std::vector<Tank>& Original_Tanks, std::vector<const Tank*>& sorted_tanks, int const left, int const mid, int const right)
{
    auto const SubVectorA = mid - left + 1;
    auto const SubVectorB = right - mid;

    //temporary vectors
    std::vector<Tank> LeftTanks;
    std::vector<Tank> RightTanks;

    //Copy data over from original list to the left list or right list
    for (int i = 0; i < SubVectorA; i++)
    {
        LeftTanks[i] = Original_Tanks[left + i];
    }

    for (int j = 0; j < SubVectorB; j++)
    {
        RightTanks[j] = Original_Tanks[mid + 1 + j];
    }

    auto IndexVectorA = 0;
    auto IndexVectorB = 0;
    int IndexOfMergedList = left;

    // Compare Health and insert the tank back into the Original List on the correct position
    // Based on their health
    while (IndexVectorA < SubVectorA && IndexVectorB < SubVectorB)
    {
        if (LeftTanks[IndexVectorA].health <= RightTanks[IndexVectorA].health)
        {
            Original_Tanks[IndexOfMergedList].compare_health(LeftTanks[IndexVectorA]);
            IndexVectorA++;
        }
        else
        {
            Original_Tanks[IndexOfMergedList].compare_health(RightTanks[IndexVectorB]);
            IndexVectorB++;
        }
    }

        //Copy elements of left to Original
        while (IndexVectorA < SubVectorA)
        {
            Original_Tanks[IndexOfMergedList].compare_health(LeftTanks[IndexVectorA]);
            IndexVectorA++;
            IndexOfMergedList++;
        }

        //Copy elements of right to Original
        while (IndexVectorB < SubVectorB)
        {
            Original_Tanks[IndexOfMergedList].compare_health(RightTanks[IndexVectorB]);
            IndexVectorB++;
            IndexOfMergedList++;
        }
}

//Recursive Function. Keep splitting until list contains one element
void Tmp10::Game::Merge_Sort_Tanks(const std::vector<Tank>& Original_Tanks, std::vector<const Tank*>& sorted_tanks, int begin, int end)
{
    if (begin >= end)
    {
        return;
    }

    auto mid = begin + (end - begin) / 2;

    //Recursive Calls (Multithreaded)
    std::thread SortThread([&] {
        Merge_Sort_Tanks(Original_Tanks, sorted_tanks, begin, mid);
        });
    Merge_Sort_Tanks(Original_Tanks, sorted_Tanks, mid + 1, end);

    SortThread.join();
    Merge(Original_Tanks, sorted_tanks, begin, mid, end);
}
```

2.3 Gebruikt algoritme 3

Hoeveel sneller?	1 minuut en 18 seconden
Screenshot code	

```
//Check tank collision and nudge tanks away from each other
for (Tank& tank : tanks)
{
    if (tank.active)
    {
        int sprite_size = 19; //Size of sprite
        int position_x = tank.position.x / sprite_size;
        int position_y = tank.position.y / sprite_size;

        for (Tank& other_tank : tanks)
        {
            if (&tank == &other_tank || !other_tank.active) continue; //Doesn't look if collision is with itself
            //Other positions
            int other_position_x = other_tank.position.x / sprite_size;
            int other_position_y = other_tank.position.y / sprite_size;

            if (other_position_x == position_x && (other_position_y - 1) < position_y < (other_position_y +1))
            {
                vec2 dir = tank.get_position() - other_tank.get_position();
                float dir_squared_len = dir.sqr_length();

                float col_squared_len = (tank.get_collision_radius() + other_tank.get_collision_radius());
                col_squared_len *= col_squared_len;

                if (dir_squared_len < col_squared_len)
                {
                    tank.push(dir.normalized(), 1.f); //Push direction normalized
                }
            }
        }
    }
}
```

3.4 Totaal



In totaal was de simulatie nu dus 4 minuten en 24 seconden sneller.

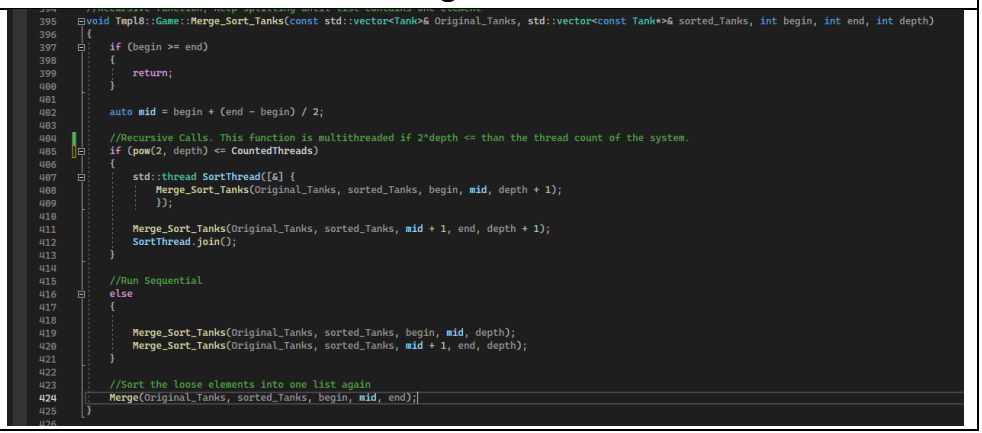
3. Concurrent

3.1 Onderzoek

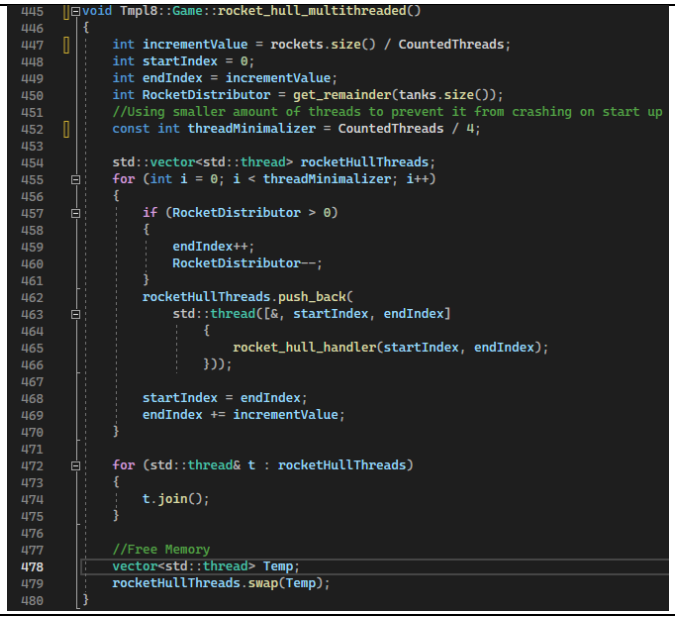
Om uit te zoeken hoe ik multithreading toe kan passen in de code, heb ik eerst de lessen van ‘Concurrent programmeren’ gevolgd.

Voor wat extra informatie heb ik het boek c++ concurrency in action (Williams, A. (2019). C++ Concurrency in Action,2E (2nd edition). New York, United States: Manning Publications).

3.2 Toepassing

Wat doet dit?	Dit is het merge sort algoritme, maar dan gemultithread. Wanneer (2^{depth} <= aantal beschikbare threads) geldt, wordt deze functie gemultithread. Anders wordt de functie sequentieel uitgevoerd.
Code	

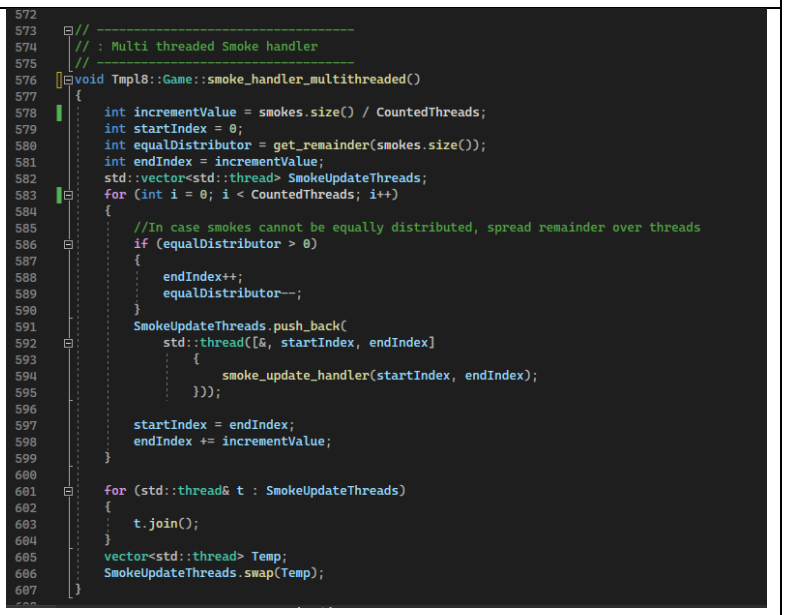
```
void Tmpl8::Game::Merge_Sort_Tanks(const std::vector<Tank>& Original_Tanks, std::vector<const Tank*>& sorted_Tanks, int begin, int end, int depth)
{
    if (begin >= end)
    {
        return;
    }

    auto mid = begin + (end - begin) / 2;

    //Recursive Calls. This function is multithreaded if 2^depth <= than the thread count of the system.
    if (pow(2, depth) <= CountedThreads)
    {
        std::thread SortThread([&] {
            Merge_Sort_Tanks(Original_Tanks, sorted_Tanks, begin, mid, depth + 1);
            });

        Merge_Sort_Tanks(Original_Tanks, sorted_Tanks, mid + 1, end, depth + 1);
        SortThread.join();
    }

    //Run Sequential
    else
    {

        Merge_Sort_Tanks(Original_Tanks, sorted_Tanks, begin, mid, depth);
        Merge_Sort_Tanks(Original_Tanks, sorted_Tanks, mid + 1, end, depth);
    }

    //Sort the loose elements into one list again
    Merge(Original_Tanks, sorted_Tanks, begin, mid, end);
}
```

Wat doet dit?	Als de raketten (rockets) het krachtveld (forcefield) benaderen, dan stopt de simulatie de raketten.
Code	

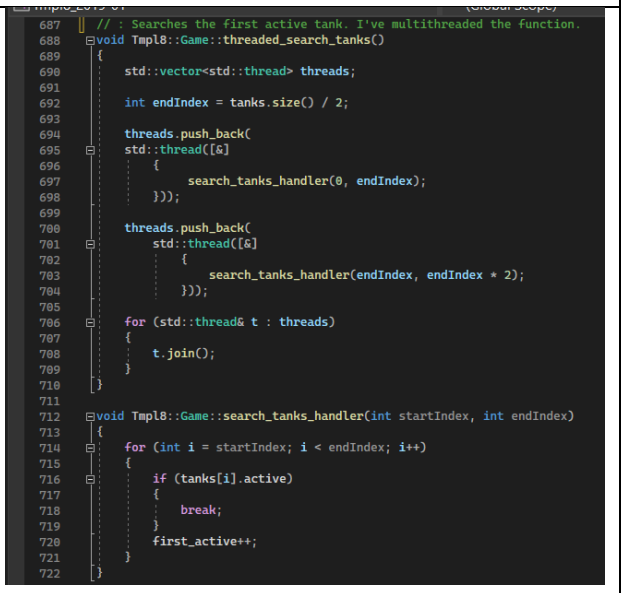
```
void Tmpl8::Game::rocket_hull_multithreaded()
{
    int incrementValue = rockets.size() / CountedThreads;
    int startIndex = 0;
    int endIndex = incrementValue;
    int RocketDistributor = get_remainder(tanks.size());
    //Using smaller amount of threads to prevent it from crashing on start up
    const int threadMinimalizer = CountedThreads / 4;

    std::vector<std::thread> rocketHullThreads;
    for (int i = 0; i < threadMinimalizer; i++)
    {
        if (RocketDistributor > 0)
        {
            endIndex++;
            RocketDistributor--;
        }
        rocketHullThreads.push_back(
            std::thread([&, startIndex, endIndex]
                {
                    rocket_hull_handler(startIndex, endIndex);
                }));

        startIndex = endIndex;
        endIndex += incrementValue;
    }

    for (std::thread& t : rocketHullThreads)
    {
        t.join();
    }

    //Free Memory
    vector<std::thread> Temp;
    rocketHullThreads.swap(Temp);
}
```

Wat doet dit?	De tank update functie is nu gemultithread.
Code	

```cpp
void Tmpl8::Game::tank_update_multithreaded()
{
    int incrementValue = tanks.size() / CountedThreads;
    int startIndex = 0;
    int endIndex = incrementValue;
    int TankDistrib = get_remainder(tanks.size());
    std::vector<std::thread> ThreadsForTankUpdate;
    for (int i = 0; i < CountedThreads; i++)
    {
        if (TankDistrib > 0)
        {
            endIndex++;
            TankDistrib--;
        }
#if 0
        if (i == threadCount - 1)
        {
            endIndex += (smokes.size() - endIndex - 1);
        }
#endif
        ThreadsForTankUpdate.push_back(
            std::thread([&, startIndex, endIndex]
                {
                    tank_update_handler(startIndex, endIndex);
                }));

        startIndex = endIndex;
        endIndex += incrementValue;
    }

    for (std::thread& t : ThreadsForTankUpdate)
    {
        t.join();
    }
```

Wat doet dit?	De smoke handler is nu ook gemultithread.
Code	

```cpp

// -----------------------------------
// : Multi threaded Smoke handler
// -----------------------------------
void Tmpl8::Game::smoke_handler_multithreaded()
{
    int incrementValue = smokes.size() / CountedThreads;
    int startIndex = 0;
    int equalDistributor = get_remainder(smokes.size());
    int endIndex = incrementValue;
    std::vector<std::thread> SmokeUpdateThreads;
    for (int i = 0; i < CountedThreads; i++)
    {
        //In case smokes cannot be equally distributed, spread remainder over threads
        if (equalDistributor > 0)
        {
            endIndex++;
            equalDistributor--;
        }
        SmokeUpdateThreads.push_back(
            std::thread([&, startIndex, endIndex]
                {
                    smoke_update_handler(startIndex, endIndex);
                }));

        startIndex = endIndex;
        endIndex += incrementValue;
    }

    for (std::thread& t : SmokeUpdateThreads)
    {
        t.join();
    }
    vector<std::thread> Temp;
    SmokeUpdateThreads.swap(Temp);
}
```

Wat doet dit?	Deze functie zoekt de eerste actieve tank. De functie is gemultithread.
Code	

```cpp
// : Searches the first active tank. I've multithreaded the function.
void Tmpl8::Game::threaded_search_tanks()
{
    std::vector<std::thread> threads;

    int endIndex = tanks.size() / 2;

    threads.push_back(
    std::thread([&]
        {
            search_tanks_handler(0, endIndex);
        }));

    threads.push_back(
        std::thread([&]
            {
                search_tanks_handler(endIndex, endIndex * 2);
            }));

    for (std::thread& t : threads)
    {
        t.join();
    }
}

void Tmpl8::Game::search_tanks_handler(int startIndex, int endIndex)
{
    for (int i = startIndex; i < endIndex; i++)
    {
        if (tanks[i].active)
        {
            break;
        }
        first_active++;
    }
}
```

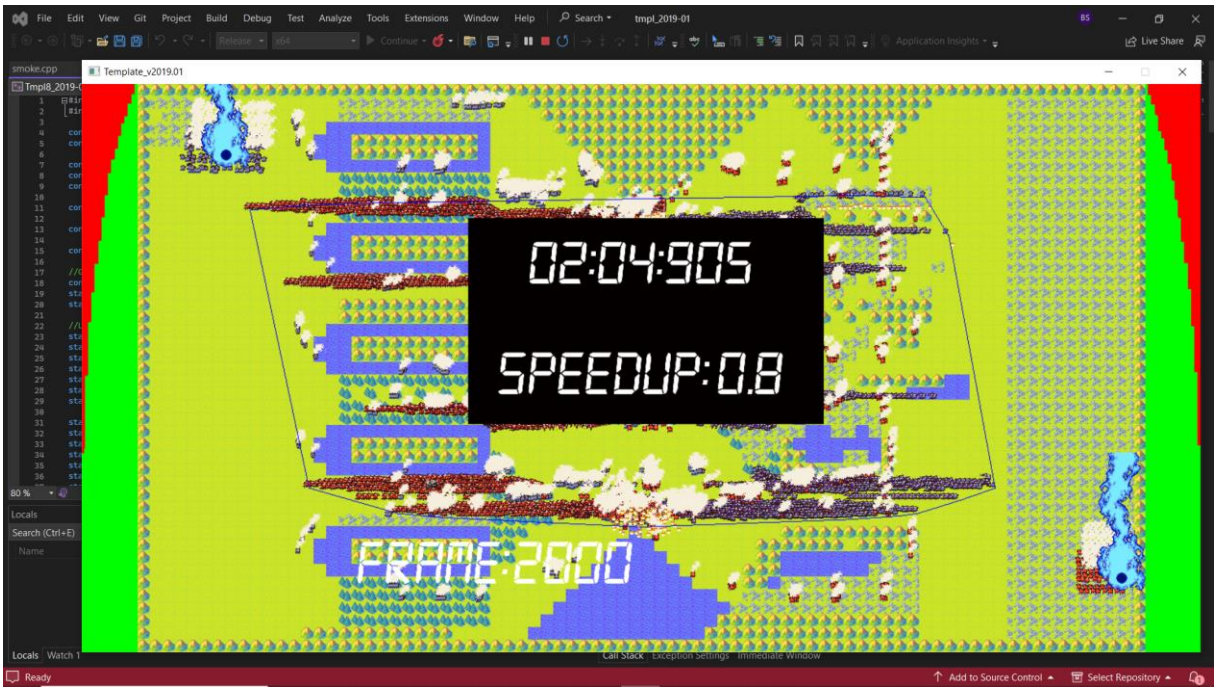
Wat doet dit?	Kijkt wanneer rockets elkaar raken. U raad het al: Deze functie is ook gemultithread.
Code	

```cpp
// Rocket collision is now also multithreaded
void Tmpl8::Game::rocket_collision_multithreaded()
{
    std::vector<std::thread> threads;
    int incrementValue = rockets.size() / CountedThreads;
    int startIndex = 0;
    int endIndex = incrementValue;
    int RocketRemainder = get_remainder(rockets.size());
    for (int i = 0; i < CountedThreads; i++)
    {
        if (RocketRemainder > 0)    //equally distribute the remainder across threads
        {
            endIndex++;
            RocketRemainder--;
        }
        threads.push_back(
            std::thread([&, startIndex, endIndex] {
                rocket_collision_handler(startIndex, endIndex);
                }));
        startIndex = endIndex;
        endIndex += incrementValue;
    }

    for (std::thread& t : threads)
    {
        t.join();
    }
    vector<std::thread> Temp; //free memory
    threads.swap(Temp);
}
```

Wat doet dit?	Explosion handler. De remainder wordt verdeeld over de threads.
Code	

```
786
787  //  Multi threaded explosions
788  void Tmpl8::Game::explosion_handler_multithreaded()
789  {
790      std::vector<std::thread> ExplosionThreads;
791      int incrementValue = explosions.size() / CountedThreads;
792      int startIndex = 0;
793      int endIndex = incrementValue;
794      int explosionRemainder = get_remainder(explosions.size());
795      for (int i = 0; i < CountedThreads; i++)
796      {
797          if (explosionRemainder > 0) //Distribute remainder across threads
798          {
799              endIndex++;
800              explosionRemainder--;
801          }
802          ExplosionThreads.push_back(
803              std::thread([&, startIndex, endIndex] {
804                  explosion_handler(startIndex, endIndex);
805                  }));
806          startIndex = endIndex;
807          endIndex += incrementValue;
808      }
809
810      for (std::thread& t : ExplosionThreads)
811      {
812          t.join();
813      }
814
815      vector<std::thread> Temp;
816      ExplosionThreads.swap(Temp);
817  }
```

3.3 Totaal



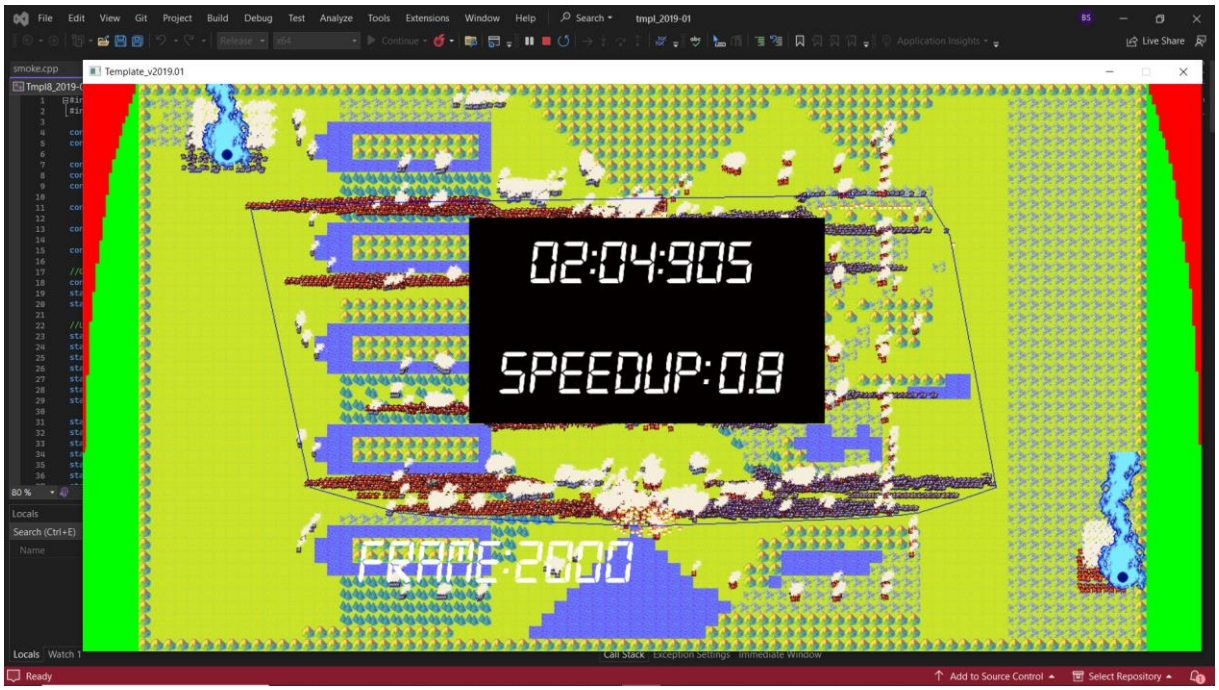
Na het multithreaden van de opdracht is de opdracht een flink stuk sneller. Maar liefst 6 minuten en 8 seconden!

4. Conclusie

4.1 Speedup

Onderdeel	Tijd	Rekensymbool
Start	12 minuten en 36 seconden	
		-
Algoritmiek	4 minuten en 24 seconden	
Concurrent	6 minuten en 8 seconden	
		=
Eind	2 minuten en 4 seconden	

Het programma deed er eerst 12 minuten en 36 seconden over om op te starten. Na alle verbeteringen duurde het nog maar 2 minuten en 4 seconden voordat het programma volledig klaar is. Dat is dus een verbetering van maar liefst 10 minuten en 32 seconden!



5. Bronnen en bijlagen

5.1 Bronnen

In APA 6e editie

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms, fourth edition (4de editie). The MIT Press.

Wikipedia-bijdragers. (2021, 16 juni). C++. Geraadpleegd op 8 januari 2024, van https://nl.wikipedia.org/wiki/C%2B%2B

Williams, A. (2019). C++ Concurrency in Action,2E (2nd edition). New York, United States: Manning Publications.