**Initial Project Plan (week 10, submission date: 26th May 2023)**

|  |  |
| --- | --- |
| **Group Name** | **🇫🇷 Baguette&Co 🥖** |
| **Members** | |  |  |  | | --- | --- | --- | | **Name** | **Email** | **Phone number** | | **Yanis Guerin Berrabeh** | **goutalyanisgoutal@gmail.com** | **+33 6 65 91 30 67** | | **Thomas Seignour** | **tseignour@gmail.com** | **+33 7 68 55 24 04** | | **Denis Aira-Benvenuti** | **denis.aira2003@gmail.com** | **+33 6 51 32 43 83** | |
| **Problem scenario description** | **Making 3D games without exploding computers because they are not strong enough to handle a lot of computations.** |
| **Why it is important** | **In the early 80s, games were only 2D. Mario, Metroid, Castlevania… Doing 3D was really hard if not impossible as computers could not handle a lot of operations. However, programmers found out about BSP, which only renders points that the player could see in front of them. Coupled with Raytracing, they could just render the closest point visible in front of the player and optimize at most computers’ performance.** |
| **Problem specification** | **We have to use the closest point algorithm, and to create our own algorithm to find which points are on the straight line using its equation.** |
| **Potential solutions** | **We want to make a program that could, given a set of points and a straight line equation, which point is on the straight line and is the closest. We can use the most efficient algorithms so computers would understand with the given closest point which point to render for one line of the raytracing.** |
| **Sketch (framework, flow, interface)** | 1. **Requirements analysis** 2. **Function 1 Input: Set of points, equation of a straight line** 3. **Function 1 output: Set of points which are on the straight line** 4. **Function 2 Input: Set of points on a straight line** 5. **Function 2 Output: The closest point of coordinate 0** 6. **Testing and debugging** 7. **Optimization and improvement** |

**Project Proposal Refinement (week 11, submission date: 2nd June 2023)**

|  |  |
| --- | --- |
| **Group Name** | **🇫🇷 Baguette&Co 🥖** |
| **Members** | |  |  | | --- | --- | | **Name** | **Role** | | **Denis Aira-Benvenuti** | **co-developer / Presentation Project** | | **Thomas Seignour** | **Main Developer / architecture program** | | **Yanis Guerin Berrabeh** | **co-developer / Project Coordinator** | |
| **Problem statement** | **Making 3D games without exploding computers because they are not strong enough to handle a lot of computations.** |
| **Objectives** | 1. **Generate a set of random points within a specified range.** 2. **Define a straight line based on a randomly generated slope value.** 3. **Identify and filter out the points that lie on the specified straight line.** 4. **Sort the points based on their x-coordinate.** 5. **Retrieve the closest point on the line from the sorted points.** 6. **Provide the coordinates of the closest point as the output.** |
| **Expected output** | **the closest point on the specified straight line from the generated set of points. We will display the x and y coordinates of the closest point.** |
| **Problem scenario description** | **Making 3D games without exploding computers because they are not strong enough to handle a lot of computations. This led to the creation of BSP technology, and our project is based on this technology and explains how the first 3D games used the closest point to display obstacles.** |
| **Why it is important** | **It's important to concentrate the power of the computer on important things** |
| **Problem specification** | **We have to use the closest point algorithm, and to create our own algorithm to find which points are on the straight line using its equation.** |
| **Potential solutions** | **We will use a Divide and Conquer (DAC) approach to solve the problem. By dividing the problem into smaller subproblems, we can efficiently filter the points, sort them, and identify the closest point. The recursive nature of DAC will help optimize the solution and provide a more efficient algorithm for finding the closest point on the line.** |
| **Sketch (framework, flow, interface)** | **1.Requirements analysis**  **2.Function 1 Input: Set of points, equation of a straight line**  **3.Function 1 output: Set of points which are on the straight line**  **4.Function 2 Input: Set of points on a straight line**  **5.Function 2 Output: The closest point of coordinate 0**  **6.Testing and debugging**  **7.Optimization and improvement** |
| **Methodology** | |  |  | | --- | --- | | **Milestone** | **Time** | | **Finding the subject of the project** | **26/5 (wk10)** | | **Defining how we will execute the project** | **2/6 (wk11)** | | **Code editing** | **9/6 (wk12)** | | **Change of paradigm** | **16/6 (wk13)** | | **Analysis of time complexity and algorthim correctness** | **16/6 (wk13)** | | **Proofreading and Powerpoint / Online portfolio** | **23/6 (wk14)** | |

**Project Progress (Week 9 – Week 14)**

|  |  |
| --- | --- |
| **Milestone 1** | **Finding the subject of the project** |
| **Date (Wk)** | **Week 10** |
| **Description/**  **sketch** | **We were looking for an idea for this project that would appeal to everyone and have practical applications today. As our group is made up of French people who have all played video games since we were children, we wondered how video games went from 2D to 3D.**  **After doing some research, we came up with the BSP (Binary Space Partitioning) technology, which was developed in the 1970s.** |

|  |  |
| --- | --- |
| **Milestone 2** | **Defining how we will execute the project** |
| **Date (Wk)** | **Week 11** |
| **Description/**  **sketch** | **This week, we started by looking at the structure we wanted to give to our programme. This stage was very complex because BSP is a complex technology that is difficult to summarise in a single code. In parallel, we looked at the type of algorithm we should choose for the BSP and our choice went to Divide and Conquer.** |

|  |  |
| --- | --- |
| **Milestone 3** | **Code editing** |
| **Date (Wk)** | **Week 12** |
| **Description/**  **sketch** | **Our code runs and we can see which parts of the programme need improving. The debugging phase is fast but we wanted to reduce the time complexity because the method used up to now has been Brute Force. This requires us to find resources online and on the Design course in order to improve time complexity.** |

|  |  |
| --- | --- |
| **Milestone 4** | **Change of paradigm / Analysis of time complexity and algorthim correctness** |
| **Date (Wk)** | **Week 13** |
| **Description/**  **sketch** | **At week 13, we found a solution to our time complexity problem by switching to Divide and Conquer. Some of the programme's functions are still in brute force and after a lot of re-examination, we were thinking of quicksort algorithms.** |

|  |  |
| --- | --- |
| **Milestone 5** | **Proofreading and Powerpoint** |
| **Date (Wk)** | **Week 14** |
| **Description/**  **sketch** | **Our programme now works and uses DAC in combination with quicksort using Lomuto partitioning. We're now focusing on creating a Git Hub and writing the project report, which took us time, read.me, algortihm correctness and the powerpoint.** |