Bilag 1

```
public static void Load_chunks(int x_off, int y_off)
{
    int[] direction = new int[2];
    //direction switch, choses direction based on offset.
    switch (x_off)
    {
        case int k when k > 0:
            direction[0] = -1;
            break;
        case int k when k < 0:
            direction[0] = 1;
            break;
        default:
            direction[0] = 0;
            break;
    }
    switch (y_off)
    {
        case int k when k > 0:
            direction[1] = 1;
            break;
        case int k when k < 0:
            direction[1] = -1;
            break;
        default:
            direction[1] = 0;
            break;
    }
    bool new_terrain = true;
    for (int i = loaded_chunks.Count-1; i >= 0; i--)
    {
        if (direction[0] + loaded_chunks[0][0] == loaded_chunks[i][0]
        && direction[1] + loaded_chunks[0][1] == loaded_chunks[i][1])
        {
            new_terrain = false;
        }
    }
    if (new_terrain)
    {
        sorter_new(direction);
    }
}
```

Bilag 2

```
public static void sorter_new(int[] new_one)
{
    switch (new_one)
    {
        case int[] n when n[0] != 0 && n[1] != 0:
            switch (new_one)
            {
                case int[] k when k[0] == -1 && k[1] == 1:
                    1
                    break;
                case int[] k when k[0] == 1 && k[1] == 1:
                    3
                    break;
                case int[] k when k[0] == -1 && k[1] == -1:
                    6
                    break;
                case int[] k when k[0] == 1 && k[1] == -1:
                    9
                    break;
            }
            break;
        case int[] n when n[0] == 1 && n[1] == 0:
            5
            break;
        case int[] n when n[0] == -1 && n[1] == 0:
            4
            break;
        case int[] n when n[0] == 0 && n[1] == 1:
            2
            break;
        case int[] n when n[0] == 0 && n[1] == -1:
            8
            break;
    }
}
```

```
#region 1
if (switch_off != 1)
{
    for (int i = loaded_chunks.Count - 1; i > 0; i--)
    {
        loaded_chunks.RemoveAt(i);
    }
    int[] adds = new int[2];
    adds[0] = loaded_chunks[0][0] + -1;
    adds[1] = loaded_chunks[0][1] + 1;
    loaded_chunks.Add(adds);
    Get_tiles(adds, 2);

    int[] adds_2 = new int[2];
    adds_2[0] = loaded_chunks[0][0] + 0;
    adds_2[1] = loaded_chunks[0][1] + 1;
    loaded_chunks.Add(adds_2);
    Get_tiles(adds_2, 3);

    int[] adds_3 = new int[2];
    adds_3[0] = loaded_chunks[0][0] + -1;
    adds_3[1] = loaded_chunks[0][1] + 0;
    loaded_chunks.Add(adds_3);
    Get_tiles(adds_3, 4);
    switch_off = 1;
}
#endregion
```

Bilag 3

```
private static void Get_tiles(int[] pos, int tile)
{
    int[] tiles = new int[width * height];
    if (chunk_check_file(pos))
    {
        tiles = Chunk_Read(pos);
    }
    else
    {
        tiles = Chunk_maker(pos);
    }
    switch (tile)
    {
        case 1:
            tiles_t_c1 = tiles;
            break;
        case 2:
            tiles_t_c2 = tiles;
            break;
        case 3:
            tiles_t_c3 = tiles;
            break;
        case 4:
            tiles_t_c4 = tiles;
            break;
    }
}
```

```
private static int[] tiles_x = new int[width * height];
private static int[] tiles_y = new int[width * height];
private static int[] tiles_t_c1 = new int[width * height];
private static int[] tiles_t_c2 = new int[width * height];
private static int[] tiles_t_c3 = new int[width * height];
private static int[] tiles_t_c4 = new int[width * height];
private static int[] tiles_empty = new int[width * height];
private static float[] tiles_mined = new float[width * height];
```

Bilag 4

```
static int[] Chunk_maker(int[] direction)
{
    int[] tiles_t = new int[width * height];
    int[] pos = new int[2];

    pos[0] = direction[0];
    pos[1] = direction[1];
    // add to list, make function to sort
    for (int i = 0; i < width * height; i++)
    {
        // function/method to see if the given x and y coordinates have a predetermined value for the terrain
        z_1 = chunk_terrain(pos, i);
        tiles_t[i] = z_1;
    }

    chunk_writer(tiles_t, pos);
    return tiles_t;
}
```

Bilag 5

```
private static void chunk_writer(int[] t, int[] direction)
{
    string filename = direction[0].ToString() + " " + direction[1].ToString() + ".txt";
    if (File.Exists(@"Chunks\" + filename))
    {
        File.Delete(@"Chunks\" + filename);
    }
    //File.CreateText(@"Chunks\" + filename); Used this earlier to create files,
    //makes the files it creates protected though, and i don't need protected files.
    string write = "";
    // loop to make the hole array to an string, adds each index to the string.
    for (int i = 0; i < t.Length; i++)
    {
        write += t[i].ToString();
        write += ",";
        if (i == 200 || i == 400)
        {
            write += "\n";
        }
    }
    // writes it.
    using (StreamWriter tw = new StreamWriter(@"Chunks\" + filename, true))
    {
        tw.WriteLine(write);
        tw.Close();
    }
}
```

Bilag 6

```
static bool chunk_check_file(int[] Position)
{
    int[] filename = new int[2];
    string negative0 = "";
    string negative1 = "";
    if (Position[0] < 0)
    {
        filename[0] = Position[0] * (-1);
        negative0 = "-";
    }
    else
    {
        filename[0] = Position[0];
    }
    if (Position[1] < 0)
    {
        filename[1] = Position[1] * (-1);
        negative1 = "-";
    }
    else
    {
        filename[1] = Position[1];
    }
    if (File.Exists(@"Chunks\" + negative0 + filename[0].ToString() + " " + negative1 + filename[1].ToString() + ".txt"))
    {
        return true;
    }
    return false;
}
```

Bilag 7

```
private static int[] Chunk_Read(int[] direction)
{
    string comma = ",";
    string current_text = "";
    int[] tiles = new int[width * height];
    int current_int = 0;
    using (StreamReader sr = new StreamReader(@"Chunks\" + direction[0].ToString() + " " + direction[1].ToString() + ".txt"))
    {
        string text = sr.ReadToEnd();
        for (int i = 0; i < text.Length; i++)
        {
            if (text[i] != comma[0])
            {
                current_text += text[i];
            }
            else
            {
                tiles[current_int] = Int32.Parse(current_text);
                current_int += 1;
                current_text = "";
            }
        }
    }

    return tiles;
}
```

Bilag 8

```
public static int Which(float x, float y, int[] chunk)
{
    int x_mod = 0;
    float y_1 = (((y / 5) - ((y / 5) % 32f)) / 32f);
    float x_1 = (((x / 5) - ((x / 5) % 32f)) / 32f);
    for (int i = 0; i < height; i++)
    {
        if (tiles_y[i * width + 1] == y_1)
        {
            x_mod = i;
            break;
        }
    }
    for (int i = 0; i < width; i++)
    {
        if (tiles_x[(x_mod * width) + i] == x_1)
        {
            for (int k = 0; k < loaded_chunks.Count; k++)
            {
                if (loaded_chunks[k][0] == chunk[0] && loaded_chunks[k][1] == chunk[1])
                {
                    switch (k)
                    {
                        case 0:
                            return tiles_t_c1[(x_mod * width) + i];
                        case 1:
                            return tiles_t_c2[(x_mod * width) + i];
                        case 2:
                            return tiles_t_c3[(x_mod * width) + i];
                        case 3:
                            return tiles_t_c4[(x_mod * width) + i];
                    }
                }
            }
        }
    }
    return 0;
}
```

Bilag 9

```
public static void Change(float x, float y, int z, int[] chunk)
{
    int x_mod = 0;
    float y_1 = (((y / 5) - ((y / 5) % 32f)) / 32f);
    float x_1 = (((x / 5) - ((x / 5) % 32f)) / 32f);
    for (int i = 0; i < height; i++)
    {
        if (tiles_y[i * width] == y_1)
        {
            x_mod = i;
            break;
        }
    }
    for (int i = 0; i < width; i++)
    {
        if (tiles_x[(x_mod * width) + i] == x_1)
        {
            for (int k = 0; k < loaded_chunks.Count; k++)
            {
                if (loaded_chunks[k][0] == chunk[0] && loaded_chunks[k][1] == chunk[1])
                {
                    switch (k)
                    {
                        case 0:
                            tiles_t_c1[(x_mod * width) + i] = z;
                            chunk_writer(tiles_t_c1, chunk);
                            break;
                        case 1:
                            tiles_t_c2[(x_mod * width) + i] = z;
                            chunk_writer(tiles_t_c2, chunk);
                            break;
                        case 2:
                            tiles_t_c3[(x_mod * width) + i] = z;
                            chunk_writer(tiles_t_c3, chunk);
                            break;
                        case 3:
                            tiles_t_c4[(x_mod * width) + i] = z;
                            chunk_writer(tiles_t_c4, chunk);
                            break;
                    }
                }
            }
        }
    }
}
```

Bilag 10

```
public static void mining_updater(float x, float y, float deltatime, int direction)
{
    int x_mod = 0;
    float y_1 = (((y) / 5) - ((y) / 5) % 32f) / 32f;
    float x_1 = (((x) / 5) - ((x) / 5) % 32f) / 32f;

    for (int i = 0; i < height; i++)
    {
        if (tiles_y[i * width] == y_1)
        {
            x_mod = i;
        }
    }
    for (int i = 0; i < width; i++)
    {
        if (tiles_x[(x_mod * width) + i] == x_1)
        {
            switch (tiles_t_c1[(x_mod * width) + i])
            {
                changing the terrain
            }

        }
    }
}
```

```
#region changing the terrain
case 2:
    if (tiles_mined[(x_mod * width) + i] > 700)
    {
        stonebreakfinish.Play();
        Terrain.Change(x, y, 1, loaded_chunks[0]);
    }
    else
    {
        mining_on_tile((x_mod * width + i), deltatime);
    }
    break;
case int n when n == 3 || n == 5:
    if (tiles_mined[(x_mod * width) + i] > 1300)
    {
        if (Terrain.Which(x, y, loaded_chunks[0]) == 5)
        {
            WorkShop.R1Cop++;
        }
        stonebreakfinish.Play();
        Terrain.Change(x, y, 1, loaded_chunks[0]);
    }
    else
    {
        mining_on_tile((x_mod * width + i), deltatime);
    }
    break;
case int n when n == 4 || n == 6:
    if (tiles_mined[(x_mod * width) + i] > 1800)
    {
        if (Terrain.Which(x, y, loaded_chunks[0]) == 6)
        {
            WorkShop.R3Tit++;
        }
        stonebreakfinish.Play();
        Terrain.Change(x, y, 1, loaded_chunks[0]);
    }
    else
    {
        mining_on_tile((x_mod * width + i), deltatime);
    }
    break;
```

```
case int n when n == 7 || n == 8:
    if (tiles_mined[(x_mod * width) + i] > 3000)
    {
        if (Terrain.Which(x, y, loaded_chunks[0]) == 6)
        {
            WorkShop.R3Tit++;
        }
        stonebreakfinish.Play();
        Terrain.Change(x, y, 1, loaded_chunks[0]);
    }
    else
    {
        mining_on_tile((x_mod * width + i), deltatime);
    }
    break;
    #endregion
```

```
private static void mining_on_tile(int i, float deltatime)
{
    if (WorkShop.Upgraded[0])
    {
        if (WorkShop.Upgraded[1])
        {
            if (WorkShop.Upgraded[2])
            {
                if (WorkShop.Upgraded[3])
                {
                    tiles_mined[i] += deltatime * 5;
                }
                else
                {
                    tiles_mined[i] += deltatime * 4;
                }
            }
            else
            {
                tiles_mined[i] += deltatime * 3;
            }
        }
        else
        {
            tiles_mined[i] += deltatime * 2;
        }
    }
    else
    {
        tiles_mined[i] += deltatime;
    }
}
```

Bilag 11

```
public static bool player_collis(int side, float deltatime)
{
    float pos_x = 0;
    float pos_y = 0;
    // 0 + amount;
    int amount_of_air_tiles = 1;
    switch (side)
    {
        case 0:
            left
            break;
        case 1:
            right
            break;
        case 2:
            up
            break;
        case 3:
            down
            break;
    }

    return false;
}
```

```
#region left
for (int j = 0; j < 2; j++)
{
    switch (j)
    {
        case 0:
            // pos check 1
            pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x;
            pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y + 1;
            break;
        case 1:
            // pos check 2
            pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x;
            pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y + 32 * 5 - 1;
            break;
    }
    // if terrain not air
    if (Terrain.Which(pos_x, pos_y, Terrain.Loaded_Chunk_differ(0)) > amount_of_air_tiles)
    {
        mining_updater(pos_x, pos_y, deltatime, 0);
        break_Sound(deltatime);
        return true;
    }
}
```

```
#region up
for (int p = 0; p < 3; p++)
{
    switch (p)
    {
        case 0:
            pos_x = 1920 / 2 - GameWorld.ofset_x;
            pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y-1;
            break;
        case 1:
            pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x + 2;
            pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y - 1;
            break;
        case 2:
            pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x + 32 * 5 - 2;
            pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y - 1;
            break;
    }
    if (Terrain.Which(pos_x, pos_y, Terrain.Loaded_Chunk_differ(0)) > amount_of_air_tiles)
    {
        mining_updater(pos_x, pos_y, deltatime, 2);
        break_Sound(deltatime);
        return true;
    }
}
#endregion
```

Bilag 12

```
public static bool player_collis_gravity()
{
    float pos_x = 0;
    float pos_y = 0;
    // 0 + amount;
    int amount_of_air_tiles = 1;
    for (int p = 0; p < 2; p++)
    {
        switch (p)
        {
            case 0:
                pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x + 2;
                pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y + 32 * 5;
                break;
            case 1:
                pos_x = 1920 / 2 - (32 * 5) / 2 - GameWorld.ofset_x + 32 * 5 - 2;
                pos_y = 1080 / 2 - (32 * 5) / 2 - GameWorld.ofset_y + 32 * 5;
                break;
        }
        if (Terrain.Which(pos_x, pos_y, Terrain.Loaded_Chunk_differ(0)) > amount_of_air_tiles)
        {
            return true;
        }
    }
    return false;
}
```

Bilag 13

```
private static void break_Sound(float deltatime)
{
    sound_timer += deltatime;
    if (sound_timer > 700)
    {
        #region rnd switch
        Random rnd = new Random();
        switch (rnd.Next(4) + 1)
        {
            case 1:
                stonebreak_1.Play();
                break;
            case 2:
                stonebreak_2.Play();
                break;
            case 3:
                stonebreak_3.Play();
                break;
            case 4:
                stonebreak_4.Play();
                break;
        }
        #endregion
        sound_timer -= 500;
    }
}
```