2IOE0 Final Report Group 1

Group 1, 2IOE0 Interactive Intelligent Systems 2019-Q2, TU/e*

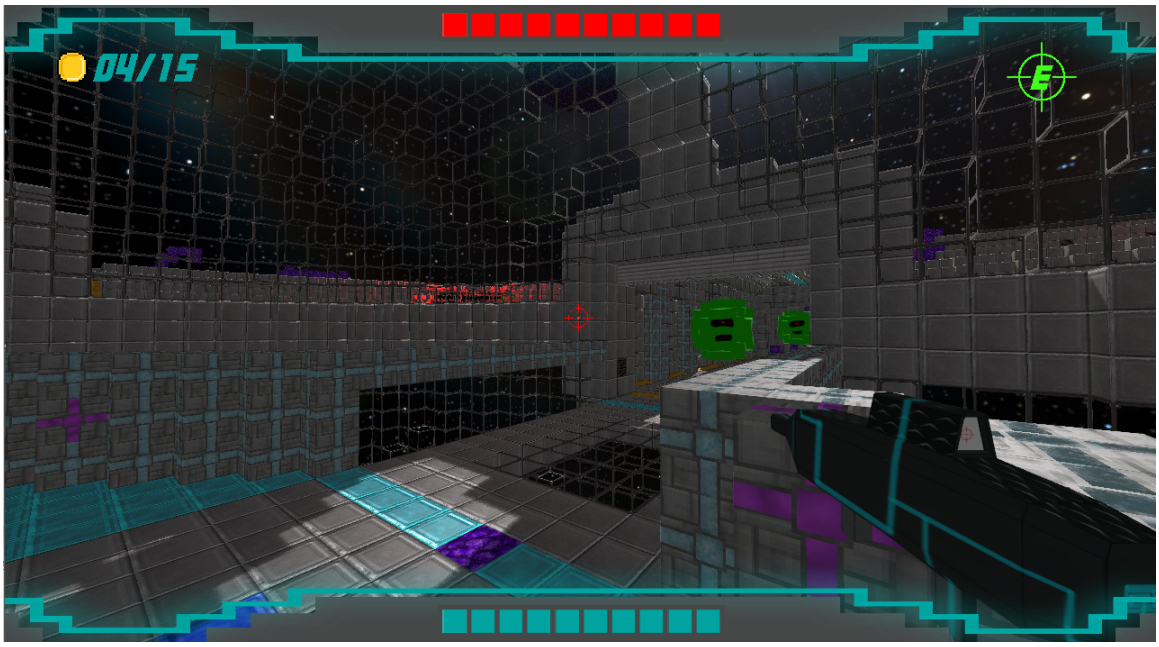

Figure 1: A screenshot of the game showcasing the visual features.

Abstract

This paper describes the researching and implementation of several Computer Graphics and Artificial Intelligence topics into a video game. The Artificial Intelligence topics are an image recognition neural network and a search algorithm, and the Computer Graphics topics which are addressed in this paper are morphing and advanced shading. These techniques were researched, and the best solutions were selected and implemented into a space-themed first-person shooter game. The neural network was implemented with a Single Shot Multibox Detector which was trained using training data generated by hand. The search algorithm was implemented with A*, as this was fast enough to run several times per second for multiple entities in the game. Morphing was done by modifying the models vertices and normals using Perlin noise, which was used because it is a continuous noise function which results in smooth movements. Finally the advanced shading was done by rendering the scene from the perspective of the light source and using the result to find what would be reached by the light, along with some filtering and noise to improve realism of the shadows.

1 Introduction

Video games are a topic in which both Computer Graphics and Artificial Intelligence are essential. This makes the creation of a video game a great method to learn about these fields, along with how to implement and test different aspects of these fields.

*Members: Daan Wöltgens, Kilian van Berlo, Mathijs Boezer, Tobin van den Hurk, Tom Verberk

The Computer Graphics topics which were researched were Advanced Shading and Morphing. Advanced Shading in this case is Shadow Mapping, which is done to increase the realism of the graphics. Shadow Mapping means that shadows are cast by a light source on objects in the world, and this must be done efficiently to keep the frame rate high and steady and therefore keep the game playable. The second topic is Morphing, which is a form of animation. Objects are modified using a time function to make the objects feel more alive.

The two Artificial Intelligence topics are neural networks and Path Finding. The Neural Network in this project is constructed to be able to recognize images of enemies in the game. The neural network does this using an image (screenshot) from the game as input, and should be fast enough to recognize the enemies within a fraction of a second after being given the image. The Path Finding is used to guide the enemies towards the player. This algorithm should be able to run multiple times in fast succession for multiple enemies in a single second, updating the paths for each of these enemies accordingly.

2 Game Design

The game is a first-person shooter with voxel-style graphics. In the game there is one playable character (PC), from whose perspective the game is viewed. The objective of this character is to collect a set of collectables (coins) which have been spread around the map at fixed locations. There are also non-playable characters (NPCs) who will attack the PC and reduce the PC's health in an attempt to stop the PC from reaching their objective. The NPCs will only spawn in specific locations on the map, at a rate that increases as the player gets closer to meeting the objective. The conflict of the game is that the NPCs are attacking the PC and thereby making it difficult to complete the objective. The map of the game will be made up of blocks with textures, as well as semi-transparent blocks like glass

Action	Control
Camera Rotation (Horizontal & Vertical)	Mouse Movement (Horizontal & Vertical)
Player Movement	W+A+S+D
Jump	Spacebar
Crouch	Ctrl
Sprint	Shift
Zoom/Aim	Right Mouse
Fire Weapon	Left Mouse
Activate Auto-Aim	E
Open/Close Pause Menu	Esc

Table 1: Player controls

blocks. The player controls the game as indicated in Table 1.

When the player actives the auto-aim ability, enemies are found by a neural network which is given by the graphical output of the game. The game then uses the positions of the enemies found by the neural network to automatically aim and fire the weapon for a short period, effectively killing all enemies currently visible to the player in one swoop.

The rules for the game are as follows:

- None of the entities can move through objects/blocks
- The PC cannot attack through objects/blocks
- The PC can attack and kill NPCs
- The NPCs can attack and kill the PC, but not other NPCs
- The PC cannot fire continuously at the same rate
- The PC cannot continuously use the auto-aim
- The NPCs spawn at marked positions on the map, at a rate proportional to the amount of collected items

There are two possible outcomes to the game. Either the player completes the objective by collecting enough of the collectables, or the PC fails in completing the objective because they succumb to the attacks of the NPCs.

2.1 Usage scenario

The game discussed in this paper does not contain a lot of different scenarios, therefore, only one scenario will be discussed which will immediately be the main one. In this scenario, you will be guided through the entire game from the first to the last screen, illustrated with Figure 15 which can be found in the appendix.

The game starts and the player sees a welcome screen, in which the player starts the game by pressing the enter key. They are then dropped in the map, where they can now start walking around on a quest to gather all the collectables. Throughout this journey hostile aliens will attack, as they want to keep the player from collecting all of the items. In the top of the screen the player's health is visible, and in the top left the progression on collecting the objectives is displayed. In order to prevent the aliens from killing them, the player can either run away or kill them by shooting them with their laser gun. However, this laser gun cannot keep firing continuously as there is a fixed amount of ammo in the gun which regenerates slower than the gun fires, this is indicated in the bottom of the screen. Having this limitation forces the player to think more strategically about when to use the gun and when not to use the gun. The more collectables they get and thus the closer they get to completing the objectives, the more aliens will spawn, which makes it harder to complete the goal. While running around and shooting aliens there are also some obstacles which the player has to either jump over,

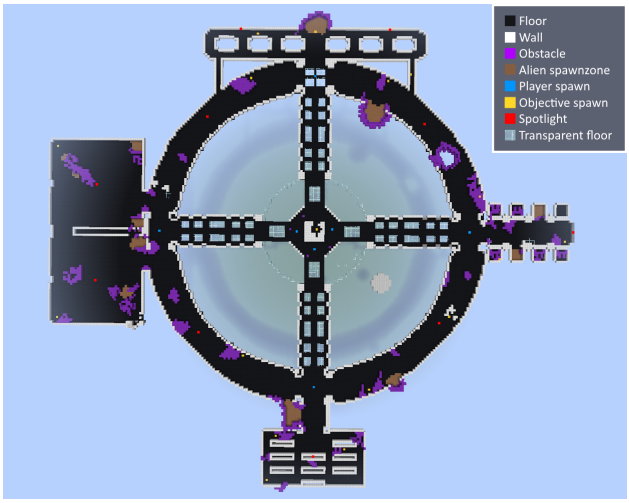

Figure 2: Top-down level design. This serves as a reference to the layout only. In Appendix B a larger picture is available as Figure 16.

crouch under or move around in order to gain an advantage over the enemies. Eventually, there are two possible outcomes; the player collects all of the collectables and wins the game or the player is killed by the aliens and loses the game.

For both cases, the options are that either the game is restarted or quit. An overview of the structure of the game that is described is given in Appendix A.

2.2 Level design

The game contains one level, to which we will refer as 'the map' (Figure 2 shows an overview). The map will represent a space station in space. This allows for a confined playing area with some scenery on the outside of the playing area. We decided to design a ring-shaped map with separate chambers on each of the sides. In the centre of the ring there is a central chamber, connected to the ring with 4 hallways. This design features obstacles to increase the difficulty and multiple intersections to allow for decision making on the side of the player. Some of the walls contain windows through which the player can see the scenery and other parts of the map. The skybox of the map will be an image of outer space. The skybox used is generated by a simulation of space in orbit of Jupiter, specifically within the vicinity of one of its moons. The PC will spawn on one of several fixed locations on the map, marked in blue. The NPC spawning areas are marked in brown, which are the points at which the NPCs can spawn. The playing area (marked in black) is confined within walls, colored white on the map. In these walls, there are some windows which the player can look through, however these are not explicitly shown on the map. The same glass object used for these windows is used in the floor in the 4 central hallways. The objectives (marked in yellow) are spread across the map and will appear on fixed locations. To increase the difficulty there are some obstacles which the player has to manoeuvre either around, over or under. These obstacles are marked purple. Although the NPC spawn points are all adjacent to these purple obstacles, they cannot cross these obstacles but instead have to move around them. The map will also contain some (colored) spotlights, which have been marked red (even though the spotlight might be a different color).

3 Conceptual Design

The game is structured around some key aspects. Two of those are from the artificial intelligence aspect: a search algorithm and object detection. The other two aspects, shadow mappin, and morphing

mainly focus on the computer graphics aspects of the game. This report mainly focuses on these aspects and how they are implemented in the game.

3.1 AI Challenges

Pathfinding
An important capability of the NPCs is that they are capable of finding their way to the PC. Given that the PC will be moving around, it is important that the defined path of the NPCs can be updated quickly. This problem is common in video game development and is known as the Moving Target Search (MTS) problem. Since the introduction of MTS in 1991 [6] there have been many algorithms based on MTS which are able to solve the problem efficiently. The map of the game is static as the map never changes and the enemies do not keep other entities in account. There are specialized algorithms which focus on moving target search, however this can also be achieved by running A* multiple times. If this is efficient enough the result is identical, which is why we decided to use this approach.

Object Detection
When the player aims and activates the special power, an AI is used to eliminate enemies (NPCs) without requiring the human player to aim themselves. Our strategy is to slow down time in the game, and take one of the frames of the game and send it to the AI. Using this approach, we need image recognition to locate NPC enemies, and then automatically eliminate all identified NPCs in an elegant manner, not by simply spraying the entire screen. For this purpose, a neural network will be used. Specifically the type of Convolutional Neural Network (CNN) is a Single Shot Multibox Detector. This network achieves a compromise between speed and accuracy compared to other similar network types. To train our network we need training data. This training data will be generated by hand by taking screenshots of the game and manually creating boxes around enemies in the screenshot. Using this training data, we will optimize the accuracy of our network such that it does not become too generous (false positives), nor too strict (false negatives).

3.2 CG Challenges

Shadow Mapping
In order to make the environment look more realistic we are going to implement Shadow Mapping. With Shadow Mapping the scene is rendered from the light's point of view. By doing this, everything we cannot see from the light's perspective is in this light's shadow and everything we can see from the light's perspective is lit. A drawback of Shadow Mapping is that there is a lot of shadow aliasing which makes the shadows less smooth. A method to reduce that problem is PCF, or Percentage Closer Filtering. PCF is a technique that takes a sample from the surrounding shadow map of a pixel and compares its depth [10]. By taking the average of the results a smoother transition takes place between light and shadow.

Animation (Morphing NPCs)
In order to make the enemies look more interesting morphing will be implemented. The NPCs will be aliens, which allow us to get creative with their design. We will be using time-dependent morphing to animate the NPCs look somewhat slimy, and alive. This will be achieved using some sort of noise which will transform the vertices and normals of the model. The noise function used will be Perlin Noise, which is a continuous noise function. This means that the movements of the model would also be continuous, which will keep the movements feeling natural, without sharp jumps.

4 MORPHING

4.1 Problem introduction

The game has a lot of obstacles and pathways that the enemies should navigate through when approaching the player. However, if we do not use any path planning algorithm for the enemies they will go straight through the walls. To make this more realistic it was decided to implement a path planning algorithm to be used by the enemies.

4.2 Possible solutions

The path planning problem has been a widely debated topic in game development and computer science as a whole. Over the years multiple algorithms have been developed and employed in order to make path finding possible for NPCs. The most famous one, and the one that will be used in the game, is A* [4]. However, more options were considered before choosing for A*. Initially we wanted to have an algorithm that could move through a dynamic environment and make the enemies able to avoid bumping into each other. Hence, we started to look at dynamic path finding algorithms and eventually found MTD* Lite [13] which seemed like the perfect solution. MTD* Lite is an algorithm that extends D* Lite [7] and uses the same principle as in Generalized Fringe-Retrieving A* [12]. It uses this principle to constantly calculate the path with the minimum cost from the enemy to the player in a dynamic environment.

According to the experiments described in the paper [13] "Moving Target D* Lite can be three to seven times faster than Generalized Adaptive A*, which so far was believed to be the fastest incremental search algorithm for solving moving target search problems in dynamic environments." When we started implementing MTD* the first step that was taken was implementing A*. When the A* algorithm was implemented we found that it actually performed well enough for our game. We also identified that MTD* Lite faced a problem in its implementation when multiple NPCs were present on one map. Multiple NPCs would make computation namely increasingly expensive. This limitation, in combination with the fact that A* was performing well enough made us decide to stick with A* instead of MTD* Lite. One extra limitation because of this though was that we were now not able to prevent enemies from colliding with each other. However, this was solved by assigning a random speed to each NPC. By doing this the NPCs will never stay in the same position as the other NPCs. Ultimately, this did not diminish the gameplay experience and therefore we decided to hold on to A* for our path planning algorithm.

4.3 How A* works

A* is one of the most used path finding algorithms and either operates on a graph or a grid (which has to be transformed to a graph). In our case, the coordinates of all accessible places on the map correspond to a certain vertex in the graph and all vertices will be connected to their neighbouring vertices in the graph by an edge. In order to calculate the shortest path from one point to another, every vertex has a predicted total cost of the path from this starting point to the prospected destination. This cost is called the F-value and is constructed from two other values of the vertex, a G-value and an H-value: $f(x) = g(x) + h(x)$. The G-value is the total value from the begin state to the current vertex. The H-value is the predicted distance from that same vertex to the destination vertex. The algorithm keeps track of two lists of vertices, the closed list and the open list. The open list is a list which is sorted and stores all the different F-values, these are the vertices that are not explored yet but for which the value has been calculated. The closed list is a list of vertices that have already been explored and is therefore not sorted.

In every iteration in A* you keep picking the vertex in the open list with the lowest F-value. For every new vertex picked, a new F-value is calculated for the vertices that are adjacent to this vertex

(if this value is lower than the current F-value). These new vertices are added to the open list with the current vertex as parent vertex. Once this is done, the current vertex is deleted from the open list and added to the closed list. You repeat this until the vertex with the lowest F-value is the destination vertex, if you find the destination vertex you can reconstruct the path using the parent vertices, which you have assigned in each iteration. You construct the path from the destination vertex to the starting vertex. The first vertex will be the destination vertex, the second vertex will be the parent vertex of the destination vertex. The next vertex will be the parent vertex from that vertex. If you continue doing this you will construct a path which will be a shortest path from the start vertex to the destination vertex.

4.4 Implementation

4.4.1 Creating the graph

When we found a possible solution for our problem we started implementing it. Since our algorithms work with graphs, the first task to be done was to translate our 3D voxel world into a graph, which could then be used by the A* algorithm. Since our map consists of voxels, we decided to let the voxels be the vertices in the graph, but that solution struck us with a new problem. Therefore we made a distinction between two types of voxels. The voxels that would be part of the graph and voxels that would not be part of the graph. We did this by looking at the y-coordinate of the voxels and the voxels above that voxel. If the voxel was at a certain y-coordinate and did not have a voxel within two meters above it, it qualified as a voxel that would be part of the graph since the enemy would be able to access this tile. Once we identified all available voxels we had a base on which we could build the graph. To know which of these voxels would be part of the final graph we had to take the width of the enemies into account. Since these do not have any collision detection with the walls except for this graph we had to make sure that the enemies did not touch the walls. Eventually we solved this by letting the voxels closest to the wall not be a vertex in our final graph. Given the width of the enemies (around one meter) this would ensure that the enemies would not come in contact with the wall. To implement this solution we created a final graph excluding these voxels (we could easily check if the eight neighbouring voxels were part of the initial graph). Thereafter we created edges between the voxels and their (horizontal and diagonal) neighbours.

4.4.2 Placing the characters in the graph

The next challenge to overcome was to make the graph useful, as we have a graph with edges representing the map. This graph has a fixed end point for each enemy, but a different starting point since they start at different positions but all have the same target, the player location. Therefore we save the vertex of the player in the graph and give the vertices of the graph different values for every enemy according to their position. The player vertex is updated every time before a new iteration of A* was run and the enemy vertex is updated every time A* is run for that particular enemy. This way the running time stays low (since the graph and vertex of the player does not have to be created every time) and the enemies kept going to the right place.

4.4.3 Using the algorithm within the game

As described before, we run A* star many times during the game to keep the enemies updated. The graph for A* is created at the beginning of the game, at which point the vertex of the player is also initialized. Once an enemy spawns it will constructs it is own A* values for the graph. A* itself is run ten times a second. This was done to keep the enemies updated for the location of the player. It was not necessary to run it more times, since their would be small to no benefit. It was also noted that A* in a short radius is not very useful anymore since there would not be any obstacles in the way

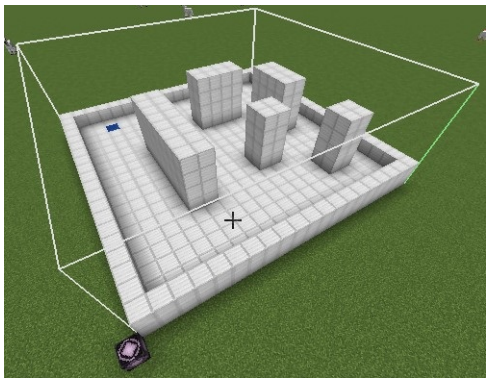

Figure 3: Test world created in Minecraft for the purpose of testing A*.

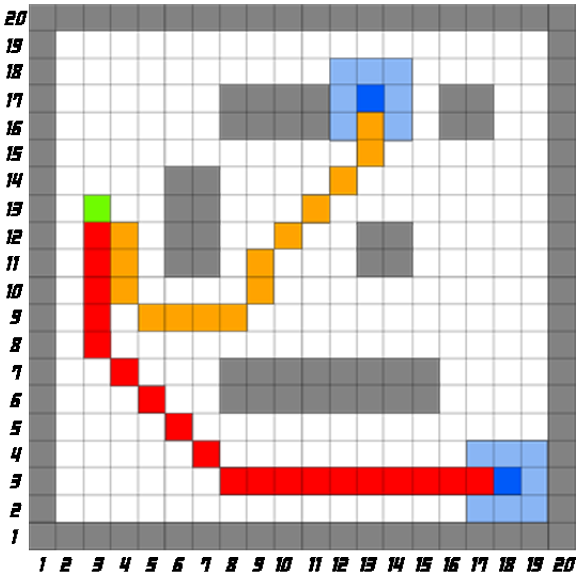

Figure 4: Visualization of the output of A* in the console.

of the enemy. Therefore we stop A* when the enemy is within two blocks of the player, the enemy then just walks towards the player, before stopping when hitting him. This is updated, in contrary to the A* algorithm, every frame.

4.5 Validation

For validating whether A* works we made the code directly output the path in the console. Since the original map we created was fairly big for testing the algorithm, we decided to create a separate 'test' map in Minecraft, as can be seen in Figure 3. This map was smaller which made it easier for us to determine what the outcome of A* should be. For testing A* we set multiple enemy positions in the map and let the algorithm find a path towards a predetermined goal position. As you can see in Figure 4 we also visualized this path in order to check the correctness of A*.

4.6 Conclusion

All in all, A* was implemented in order to make the enemies navigate through the environment in a logical manner. They follow a shortest path from their position to the player position. Its implementation does still have some limitations though. The A* for one enemy does not take into account the other enemies while moving, hence they still collide with each other. However, this seldom happens since

the starting positions are determined at random and NPCs vary in their movement speed. Another limitation of the implementation is that sometimes enemies still scour the edges of the blocks and do not take the path a person would take because of the way the edges are placed within the game. Despite these few minor limitations, A* performs well in the game.

5 Object Detection

5.1 Problem introduction

One of the mechanics in the game requires the ability to find the location of enemies within the visible view using a solution based on artificial intelligence. The essence of the problem is the following: given a frame of the game in full resolution including colors and everything the player is able to see, find the pixel locations of the enemies that are visible. The game will then use these pixel locations to calculate the direction relative to the player, which is used to send bullets in that direction as part of the game mechanic that this technique is used for.

It is also required that the detection can be done very quickly, because both the player and the enemies are in motion, and if there is a significant delay, the directions that are found will not be accurate.

5.2 Possible solutions

This gives us an object detection problem. There are several existing solutions. For a slightly simpler version of the problem, where we just want to detect whether or not a specific object is present in a picture, convolutional neural networks can be used. There also exist methods for using such a network for object detection by using another network to find regions of interest in the input image. Here the input image is first looked at by a neural network which finds regions or boxes of interest within the image, and then the second network tries to recognize what is visible within these regions. Such methods are referred to as region proposal methods. Examples of concrete solutions in this class are R-CNN [3], Fast R-CNN [2] and Faster R-CNN [11]. These solutions typically have a high accuracy, but a low speed.

Another commonly used method is You Only Look Once (YOLO) [9]. Whereas most object detection solutions use solutions that come in two parts, one part for finding regions of interest, and the other for detecting objects within the regions, YOLO only uses one neural network. This is where the name comes from, it only looks at the image once with a single neural network. This network divides the image into regions and predicts probabilities based on detections within these regions. YOLO is significantly faster compared to region proposal methods, but the accuracy is also lower.

Another method is a Single Shot MultiBox Detector (SSD) [8]. This method gives a better compromise between speed and accuracy. This method is slower than YOLO, but still significantly faster than region proposal methods. Similarly, the accuracy lies between the two, better than YOLO, but worse than region proposal methods. Like YOLO, SSD only looks at the image once. It uses a convolutional network on the input image to generate a feature map. On this feature map, detection is done, but also more convolution is done to create a new smaller feature map (so the feature represents bigger parts of the image), which is again used for detection. This process is iterated several times, each time bigger objects in the image will be detected. In the end the network gives a fixed number of detection boxes, typically represented by the image coordinates of a corner of the box, the sides of the box, the detected label and an indicator of certainty. No matter how many actual objects are in the input image, it will always find a fixed number of boxes, so action has to be taken to ignore boxes that do not actually detect real objects. To do this a cut-off value may be chosen to ignore all boxes with a certainty lower than a specific value.

This cut-off value is a compromise between false positives and false negatives. If the value is very low, everything that vaguely resembles an object will be detected, but if it is very high only those the model is very certain about will be detected. A middle way is found through experimentation.

SSD is the method chosen for solving the object detection problem. It is a good compromise between accuracy and speed.

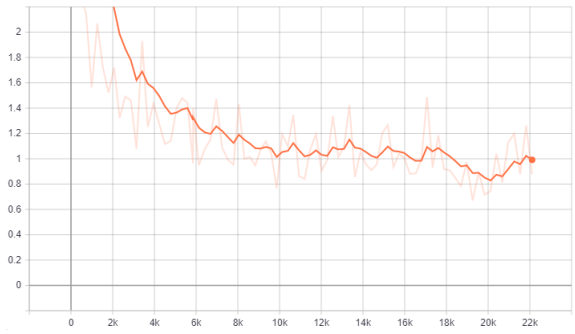

Figure 5: Loss of the model during training. Vertical axis is the loss, horizontal axis the number of steps. Note that smoothing is applied, the partially transparent line is the true loss, while the opaque line is using smoothing to reduce the effect of random noise.

5.3 Data collection

For our chosen method, SSD, it is possible to use an already trained model, and use transfer learning to train it to detect the objects that we need it to detect. This means that we need significantly less data to train a model satisfying our requirements, while also reducing the training time. To this end, a SSD model named SSD-MobileNet [5] was used as the pretrained model.

For the data, screenshots of the game were used. One hundred screenshots were taken, each containing various number of enemies. It was ensured that enemies were observed from various angles, heights and lighting conditions, to make sure that all usage scenarios were covered in the data. These one hundred screenshots were then labeled by manually creating a box around the enemies within the screenshots. Next the data was split between ninety images for training and ten images for validation. Finally the images with the box locations were converted to TensorFlow records, to be used for the training of the model.

5.4 Training the model

Tensorflow is used for training the model. in Figure 5 the loss per step count can be seen. When using transfer learning on the SSD-MobileNet model, it is recommended to try to get the loss between one and two, for reasonable performance. As can be seen, this was achieved quite rapidly. Ultimately after approximately two hours of training and over twenty thousand training steps, the training was halted, as the model was not noticeably improving anymore.

5.5 Testing the trained model

To test the model, we created some new screenshots from the game, and used the model to see how well it can predict the bounding boxes on those screenshots. In Figure 6 it can be seen that the model accurately finds the location of the enemy within the picture and finds a proper bounding box around the enemy. It also has a very high certainty, which is impressive. In Figure 7 another example is shown. This image includes two enemies somewhat further away compared to the other figure, one of which is on the outside and partially obscured by glass. The model is still able to accurately find the enemies and draw boxes on them, again with a high accuracy.

Figure 8 shows that it struggles with larger numbers of enemies. The confidence levels are significantly lower compared to the other images, and it makes a number of mistakes as well. It occasionally

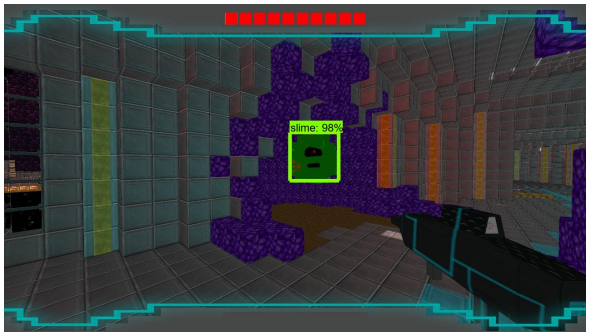

Figure 6: A screenshot of the game with a visualization of the predicted boxes from the trained model using a cut-off value of 0.1. This image shows that it accurately finds the proper box surrounding the enemy.

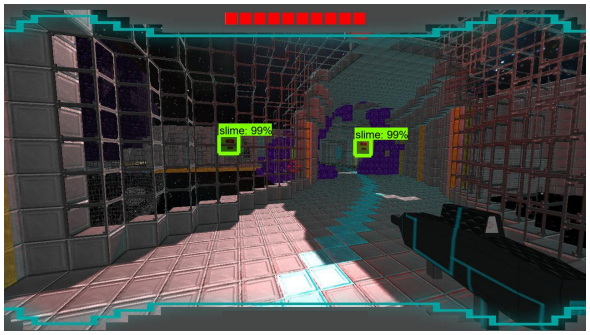

Figure 7: A screenshot of the game with a visualization of the predicted boxes from the trained model using a cut-off value of 0.1. This image shows it can accurately find multiple enemies, even when an enemy is partially obscured by glass blocks.

fails to see that two enemies close together are not one bigger enemy, and even draws two boxes on a single enemy in the right bottom corner.

5.6 Using the model within the game

TensorFlow has libraries for various programming languages, including Java, the language used for our game, hence opening the possibility of directly using the model within the game. To actually use the model directly within the game, we make it so we can extract an image from the frame buffer, which is then passed on to the model. Next the model gives us the prediction values, which converted to the direction in game using the inverse of the projection matrix, and finally bullets are send towards the detected enemies.

5.7 Validation

To validate the complete solution to the problem, we implemented a method within our game to indicate where it has found enemies. This way we can manually extract a screenshot and check if the target positions that are found are accurate, hence indicating that it is capable of properly finding the enemy locations. Such a screenshot is shown in Figure 9. It shows that it can accurately find the enemies, as was required for the game mechanic.

We also measured the duration of running object detection on a single image. This took approximately seventy milliseconds on average, however it varies depending on the resolution of the game and the hardware used. On modern computer it can be expected to be below one hundred milliseconds on Full HD resolution, which fits our requirements.

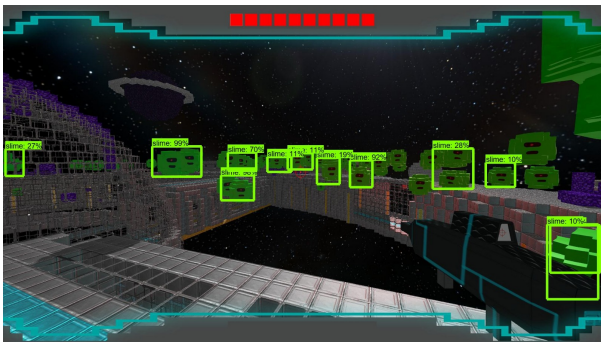

Figure 8: A screenshot of the game with a visualization of the predicted boxes from the trained model using a cut-off value of 0.1. This image shows it struggles with having large amounts of enemies in one screenshot, and when there are enemies that overlap one another.

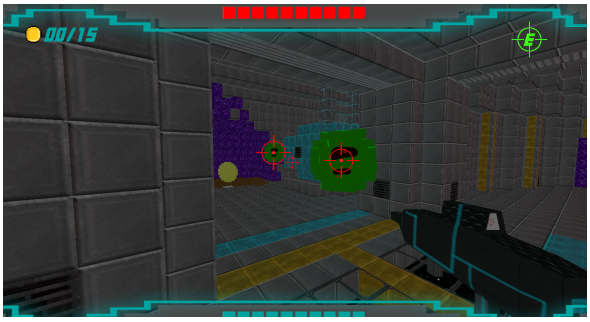

Figure 9: A screenshot of the game where it puts a crosshair on the positions it believes an enemy to be. This screenshot shows that it can accurately find the enemies within the game, as is required for the game mechanic that uses the object detection model. Note that the smaller crosshair in the center of the screen is the crosshair for the player's gun, only the bigger two crosshairs are for the enemy detection.

5.8 Conclusion

Training a model based on SSD using TensorFlow and a self-made dataset consisting of one hundred labeled images, we were able to achieve the required speed and accuracy to be able to implement the game mechanic that uses object detection for shooting enemies.

6 Shadow Mapping

6.1 Problem introduction

Light is undoubtedly the most important part of rendering a scene that looks visually appealing. A large part of that is also the absence of light. In this section we discuss how to include shadows into the game.

The game map is largely indoors, but the map also contains a significant amount of windows, letting in sunlight. Because of these circumstances, having no shadows would be very obvious. The light from the sun would go through the walls and there would be no proper shadows. Having a way to have realistic shadows would solve this issue and greatly improve the visual quality of the game.

6.2 Methods for rendering shadows

One of the best method for rendering shadows is through ray tracing [14]. At a first glance it is fairy simple, and in theory it should be almost entirely realistic. Ray tracing works by tracing the light rays between a light source and the camera. The paths of the rays are simulated as they bounce between objects and do all the interactions that light does in the real world. This method is highly realistic, but has a very high computational cost, making it impossible to use in our game while keeping the game playable on common hardware.

Another method is shadow volume [1]. This technique is not as realistic as ray tracing, but it has less computational cost. The method creates a volume for each light source that lies within the shadow the light source casts. It does this by racing rays from the light source to find a surface it shines upon. Anything behind this surface, is in the shadow volume, as the light does not go through surfaces, this must be in the shade (for that light source). This method is still fairly costly, and on top of that is quite complicated to implement.

The last method we will discuss, and the one we are using, is Shadow Mapping [15]. The basic principle behind Shadow Mapping is very simple. A light source can never see its own shadow, because anything in the shade must not have a direct line of sight to the light source. Therefore if we render a scene from the perspective of the light source, anything that can be seen is in the light, and anything besides that is in the shade. By optimizing away things like textures, and decreasing the resolution, this render of the scene can be made quite efficient, therefore making this method cost efficient. The realism of the method is decent, although the shadow map resolution is an important factor as to how realistic the shadow is.

6.3 Implementation

We implemented Shadow Mapping by rendering the scene from the perspective of the sun. Since the sun can be considered to be infinitely far away for lighting purposes, we render the scene using an orthogonal projection. Not the entire scenes needs to be rendered though, just the parts of the scene that need to cast shadows. This allows us to exclude most of what lies behind the camera. Some things still need to be rendered, since the camera might have the sun behind itself and an object blocking the sun also behind itself. In this scenario the camera should see the shadow of an object that is not in its perspective. To ensure this looks right, consideration was taken to include such objects while still keeping the volume of the scene low, to reduce computational costs.

To find whether a fragment is within the shade or not, we must know where this fragment is in the perspective of the light source, therefore we need to have a matrix to calculate this position. Then it can compare the distance from the fragment to the light source with the shadow map. This shadow map contains information about the distance of the closest parts to the light source, so it can easily check if it is the closest, in which case it is in the light, or not, in which case it is in the shade.

Consideration must be taken for objects that are not in the shade. Due to rounding errors inherent in how numbers are represented, it could be that an object casts a shadow on it self. For example, for some fragment it will calculate that it is x distance units from the light source, and for that location the closest object to the light source is also x distance units from the light source (because it is outside the shade). This would mean that it is in the light, however due to rounding errors, the values for x which should be the same, may differ slightly. To fix this an offset is used to account for these rounding errors.

6.4 Dynamic resolution

The shadow map is one single map with a certain resolution. This is evenly spread throughout the perspective of the light source. This means that shadows that are on the edge of the render distance for shadows are as sharp as those right next to the camera. With a sufficiently high resolution, this does not matter. However, having a high resolution makes rendering more expensive, hence it is desired to keep the resolution as low as possible.

A solution is to render the scene several times at different distances. One render may be made for the first few meters from the camera, another for a bit further, and a last one for the complete shadow distance. This way the shadows closer to the player will be sharper, while those further away will be more blunt, but this does not matter, because these take up a smaller part of the screen.

This method is analogous to mipmapping [16]. In mipmapping similarly different densities of pixels are used for rendering, although for mipmapping the resolution is adjusted, while for Shadow Mapping we have chosen to vary the size of the shadow map in world space.

6.5 Percentage Closer Filtering

In Figure 10 you can see what the previously mentioned gives us so far. The shadow is fairly accurate in terms of geometry, but the pixelation is obvious. Besides the pixelation, it is also very shard. There is no transition between the shade and the light. Real shadows are rarely as sharp as seen in the Figure. There should be a transition between the parts in the shade and those outside the shade. This, and the reduction of pixelation, can be achieved using Percentage Closer Filtering (PCF).

PCF changes how the shade of a fragment is calculated. Instead of just checking if that fragment is in the shade or not, it checks a region around itself. For parts entirely in the shade, the entire region will be equally shaded, similarly for parts entirely outside of the shade. For parts on a transition however, some percentage of the region will be shaded. This percentage can be used for calculating the shade instead.

This method results in what can be seen in Figure 11. You can clearly see that shadows now contain a transition area. It is still very obvious that shadows exist out of pixels though.

6.6 Adding a noise factor

To further increase the realism of our Shadow Mapping, another modification was made to how the shade is calculated. Instead of checking a region around itself, the fragment now also uses a random noise factor. This makes it so that this region is not strictly defined to be a square around itself, but may have some variance. This gives the result what can be seen in Figure 12. This hides the pixelation of the shadows and further increases the realism. It does introduce some visible grainy noise onto the shadow transitions though, but with the proper parameters this can be kept to a minimum.

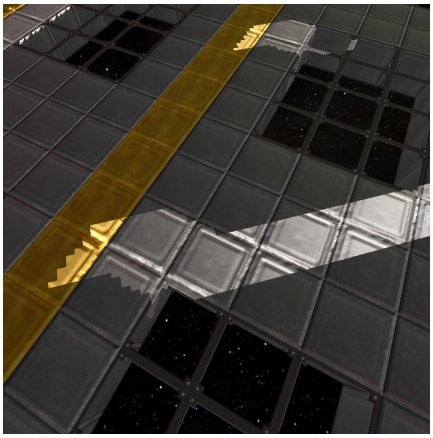

Figure 10: An example of how the shadows look with basic Shadow Mapping. The resolution has been decreased to put emphasis on the pixelation.

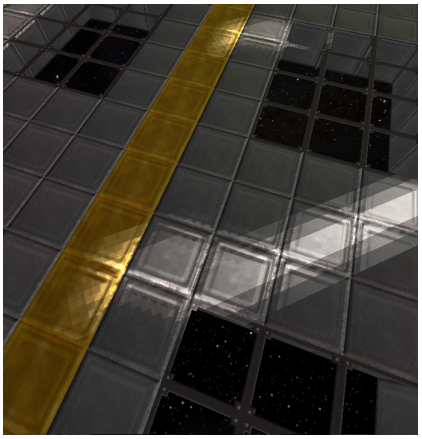

Figure 11: An example of how the shadows look with basic Shadow Mapping and PCF. The resolution has been decreased to put emphasis on the pixelation.

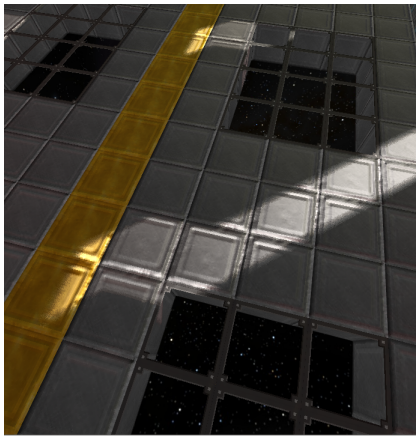

Figure 12: An example of how the shadows look with basic Shadow Mapping and PCF with noise.

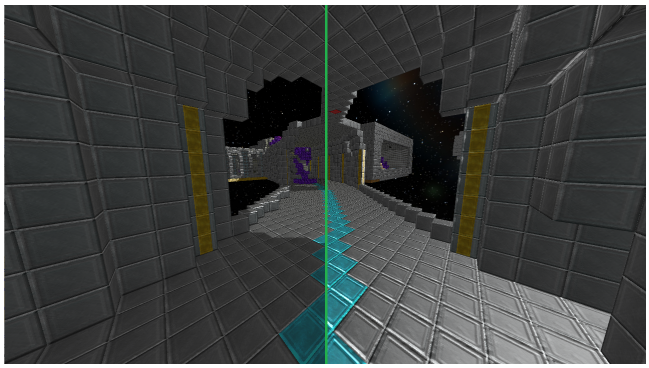

Figure 13: A comparison of a scene with and without our Shadow Mapping solution. Left is with Shadow Mapping, right is without. On the left shadows are visible, while on the right parts in shade are still lit as though they are in the sun.

6.7 Validation

In Figure 13 a scene can be seen with and without Shadow Mapping, side by side. It is clear that the sun casts proper shadows now. This greatly increase the visuals of the game.

The performance has decreased somewhat as a result of this implementation. The average frame rate without Shadow Mapping is 1220 frames per second while with Shadow Mapping it is 650 frames per second on a particular hardware set up, with the resolution and scene being equal. This is a significant decrease, however it is acceptable considering the increase in the visual quality of the game.

6.8 Conclusion

To conclude, an implementation was made to create shadows within the game. Through various methods these shadows can be made quite realistic. It does have its limitations though. Adapting Shadow Mapping for point lights rather than a light source like the sun is significantly more complex. Besides that, the shadows are still not quite perfect in terms of realism, particularly at the edges some artifacts may be seen when the light source is rotating.

7 Morphing NPCs

7.1 Problem introduction

To make the enemies more interesting, it was decided to introduce a morphing animation into the enemy model. With morphing animation is meant some animation that makes it seem like it is continuously changing in some randomized manner by changing the distance of a vertex of the model to the center of the model. Each vertex should have an independent increase or decrease of this distance, so it is not acceptable to simply increase and decrease the scale of the model, which would be the case if each vertex was increased by the same amount. This morphing animation must also be continuous, or in practice, seem continuous, since the game is of course based on discrete calculations. This continuity is both in the shape of the model, so it must always be the case that two adjacent vertices have almost the same offset, for example. It must also be continuous with regards to the time, the offsets should not change much between t and $t+\varepsilon$ for some small ε.

From this we can get the proper problem statement. Given the position of a vertex and a variable representing the time, calculate the offset to the position of the vertex to satisfy the previously mentioned properties.

7.2 Possible solutions

This problem statement comes down to finding some sort of noise function that is continuous in the manner previously described. What

we need is a four dimensional noise function. It needs three dimensions for the input location of the vertex and one dimension for the time. This means we need a noise function that takes 4 inputs and gives one output in the range between zero and one.

There are several options for these requirements. A few of the commonly used ones are Simplex noise, OpenSimplex noise and Perlin noise. All of these satisfy the requirements and have approximately the same computational requirements. For the game it was decided to use Perlin noise, because one team member has previous experience with Perlin noise. Other noise functions would have sufficed as well though.

7.3 Normal adjustments

The normal vectors of modified vertices are significantly harder to calculate. They need to be changed, because otherwise the lighting stays the same, meaning from certain angles it is not even obvious anything is changing at all.

While it is possible to try to get a gradient from the Perlin noise function itself, even this would not be a satisfactory solution yet, because you would still need to take into account the geometry of the model for finding the proper adjusted normal vectors.

Instead of taking that approach, we did something simpler. Instead of calculating the true normal vectors, we just change the original normal vectors from the model data slightly. This is not accurate, however it looks accurate unless you know what you are looking for. This method gives a very simple approximation method for getting roughly the same result. While it will probably look somewhat different from the true normals, it is not something you would notice without deliberately looking for flaws in the shading of the morphed models.

7.4 The implementation

For modifying the vertex locations, it is needed to calculate the Perlin noise value in the vertex shader. This means that we need an implementation of the four dimensional perlin noise function inside of the shader. Luckily there were existing implementations of this function within the shader language.

Using this function, we could quite easily use the normalized vertex location with a time uniform to calculate the phase of the offset. Together with some constants for the amplitude and density of the noise, the morphing was implemented. For the normals we slightly rotate the existing normals according to the offset from the Perlin noise. As mentioned before this is not realistic, but it looks realistic enough for the purposes of our game.

To make sure that the enemies are not all in sync, each enemy takes a random number as the starting point for the time parameter of the Perlin noise function.

7.5 Validation

In Figure 14 you can see four phases of an enemy. Here it is visible how the enemy deforms while still approximately retraining its original shape. The difference in lighting is also visible. It looks realistic, but if one were to freeze the time parameter of the noise function and to manually trace the light ray, it would become clear that the normals are actually not accurate, even though they look that way.

7.6 Conclusion

In conclusion, using Perlin noise gives a satisfactory method of morphing the enemy model in such a way that it look interesting, while still retaining its original shape. The computational cost of morphing is neglectable, because the computation itself is not significant, and everything is done in the vertex shaders, rather than the fragment shaders.

Figure 14: Four phases of the slime morphing animation

8 Other features

Besides the computer graphics and artificial intelligence features discussed in this paper, there were some other interesting features added to the game. These features will be discussed in this section.

8.1 Moving Skybox

As discussed in the Shadow Mapping chapter of this report we have implemented some advanced Shadow Mapping in our game. Shadow Mapping is implemented to support a moving light source, like the sun. This means that we have the ability to have a moving sun. The sun was decided to be part of the skybox. Since everything is in space, this gives the effect of the space station itself rotation around its own axis, rather than the rest of the world rotating around it. This makes the world feel more alive, and the shadows a lot more interesting, since they move.

8.2 Spotlights

To improve the feel of the game, spotlights were implemented. These lights are an extra source of light next to the sun. The lights are often placed in dark places of the map, this to ensure that there is sufficient light for the player to see. The colors of the lights are configurable and several are used. An example are the yellow lights, which are indicators for a route to the outside. The lights vary over time in brightness, which lets us configure some lights to be blinking. This is used to attract attention to the light, or to make them look like alarm lights. Overall we are satisfied with how these spotlights turned out.

8.3 Normal mapping

One of the techniques used to make graphics look more real are so called normal maps. Normal mapping is used to change the normals of a flat surface of a voxel to make it look as though it has a complex geometry, making the voxels look more realistic without having to use a complex model.

8.4 Collision Detection

The world has a fairly complex geometry, and all of it is available to the player. The player is not supposed to be able to walk through walls, fall through floors and so on, so it does not suffice to let

the player have some fixed vertical level. We need to have proper collision detection with each voxel in the world.

To implement this a KD-tree is used to store all the collision boxes for each voxel. Every time the player moves this is used to calculate the proper interactions. This lets the player interact with the world, like walking against walls, walking on different levels and so on, without compromising too much of the performance.

9 Results

The game was created according to the earlier specified game design. The results of the various features of the game have been discussed in earlier sections. Now the overall result of the game itself will be looked at.

Since it is a game, it must be playable, and to be playable, it must have perform well on reasonable hardware. To be specific, the framerate must be sufficiently high to allow the player to play the game.

We compare two hardware set ups. In both cases a scenario is used where a significant part of the world is on the screen, to represent a worst case scenario. Positioning the player in such a way that very little of the map is visible significantly increases the framerate, however this is not representative of a realistic play scenario.

Our game runs at approximately 75 frames per second on a NVIDIA Quadro M1200M graphics card with an Intel Core i7-7700HQ processor. With this hardware, the graphics card is the bottleneck. This entails that the processor is waiting on the graphics card most of the time.

Our game runs at approximately 700 frames per second on a NVIDIA RTX2070 graphics card with an Intel Core i7-4770k processor. Here too, the graphics card is the bottleneck.

This shows that the processor is not being utilized by any significant amount. Rendering the scene takes up most of the time, even on a powerful graphics card like on our second hardware set up. Because of this, performance optimizations on the side of the processor would have very little purpose, because it is not the current bottleneck.

Finally we observe the visual quality of the game. This is highly subjective. The reader can observe the game in Figure 1. It is visible that there is a significant amount of detail, with moderately realistic lighting. The aim of the game is not realism however, but merely to be visually pleasing, which we consider to be sufficiently accomplished.

10 Conclusion

In this project two Computer Graphics topics and two Artificial Intelligence topics were researched and implemented into a game. The Computer Graphics topics were Advanced Shading and Morphing, and the Artificial Intelligence topics were Neural Networks and Path Finding.

The first topic discussed was path finding. The simplest solution is A*, as this is a very simple algorithm to implement, another option considered was MTD* Lite which is a much more complex algorithm. The world was converted into a graph representation, and through testing it was found that A* was already efficient enough to run on all of the NPCs ten times a second. Therefore we decided to keep this as the implementation.

The second topic researched was Object Detection. This could be accomplished using a neural network, so several neural network methods were considered. These included R-CNN, Fast R-CNN, Faster R-CNN, You Only Look Once (YOLO) and Single Shot Multibox Detector (SSD). The decision was eventually made to use the SSD as this method would give a compromise between speed and accuracy of the results. A dataset was made using one hundred screenshots from the game and an existing model, the SSD-MobileNet, was retrained using the data. Finally the model was tested and validated, which showed that the neural network was able to recognize the NPCs within approximately seventy milliseconds. This is fast enough for the model to be useful during the game.

The third topic was advanced shading, and specifically shadow rendering. A few different methods were researched for this topic, including ray tracing, shadow volume, and Shadow Mapping. The most efficient method of rendering, and also the simplest to implement is Shadow Mapping, which is why it was decided to use this method, which is done by rendering the scene from the perspective of the light source. The shadow is then also filtered and noise is added to make the edges of the shadows less sharp. For efficiency three different resolution maps are made so that only areas close to the player are rendered very detailed. This resulted in realistic shadows which were dynamic, however the shadows are only cast for one light source, for more it would become inefficient.

The final topic researched was Morphing which was used to make the NPCs more interesting, by adding some animation to the models. The normals of the model would need to move, and the movement had to be continuous to keep the animations smooth. There were several noise functions considered, but eventually Perlin Noise was selected as the function which would be used. The vertices of the NPC models were then altered by the noise, and the normals of the model were approximated. The performance of the game was not significantly affected by this feature, and the NPCs are made more interesting by the feature.

In the game all of the topics which were researched were able to be implemented. The game is able to run smoothly which means that the features which were implemented were also implemented efficiently.

References

[1] F. C. Crow. SHADOW ALGORITHMS FOR COMPUTER GRAPHICS. *Computer Graphics*, 11(2):242–248, 1977. doi: 10.1016/b978-155860651-7/50154-6

[2] R. Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015:1440–1448, 2015. doi: 10.1109/ICCV.2015.169

[3] R. Girshick, J. Donahue, T. Darrell, J. Malik, U. C. Berkeley, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:5000, 2014. doi: 10.1109/CVPR.2014.81

[4] P. E. Hart, N. J. Nilsson, and B. Raphael. Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics*, 4(2):100–107, 1968.

[5] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2017.

[6] T. Ishida and R. E. Korf. Moving Target Search. *International Joint Conference on Artificial Intelligence (IJCAI)*, (x):204–210, 1991.

[7] S. Koenig and M. Likhachev. D* Lite. *Proceedings of the National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016. doi: 10.1007/978-3-319-46448-0_2

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:779–788, 2016. doi: 10.1109/CVPR.2016.91

[10] W. T. Reeves, D. H. Salesinf, and R. L. Cook. Rendering antialiased shadows with depth maps. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, 21(4):283–291, 1987. doi: 10.1145/37401.37435

[11] J. Ren, S., He, K., Girshick, R., & Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in neural information processing systems*, pp. 91–99, 2015. doi: 10.2307/j.ctt1d98bxx.10

[12] X. Sun, W. Yeoh, and S. Koenig. Generalized fringe-retrieving A*: Faster moving target search on state lattices. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2:1081–1087, 2010.

[13] X. Sun, W. Yeoh, and S. Koenig. Moving target D* Lite*. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 1(Aamas):67–74, 2010.

[14] T. Whitted. An improved illumination model for shaded display. *ACM SIGGRAPH 2005 Courses, SIGGRAPH 2005*, 23(6), 2005. doi: 10.1145/1198555.1198743

[15] L. Williams. CASTING CURVED SHADOWS ON CURVED SURFACES. *Proceedings of the ACM SIGGRAPH*, pp. 270–274, 1978. doi: 10.1017/CBO9781107415324.004

[16] L. Williams. Pyramidal Parametrics. *Computer Graphics*, 17(3), 1983.

A GAME STRUCTURE

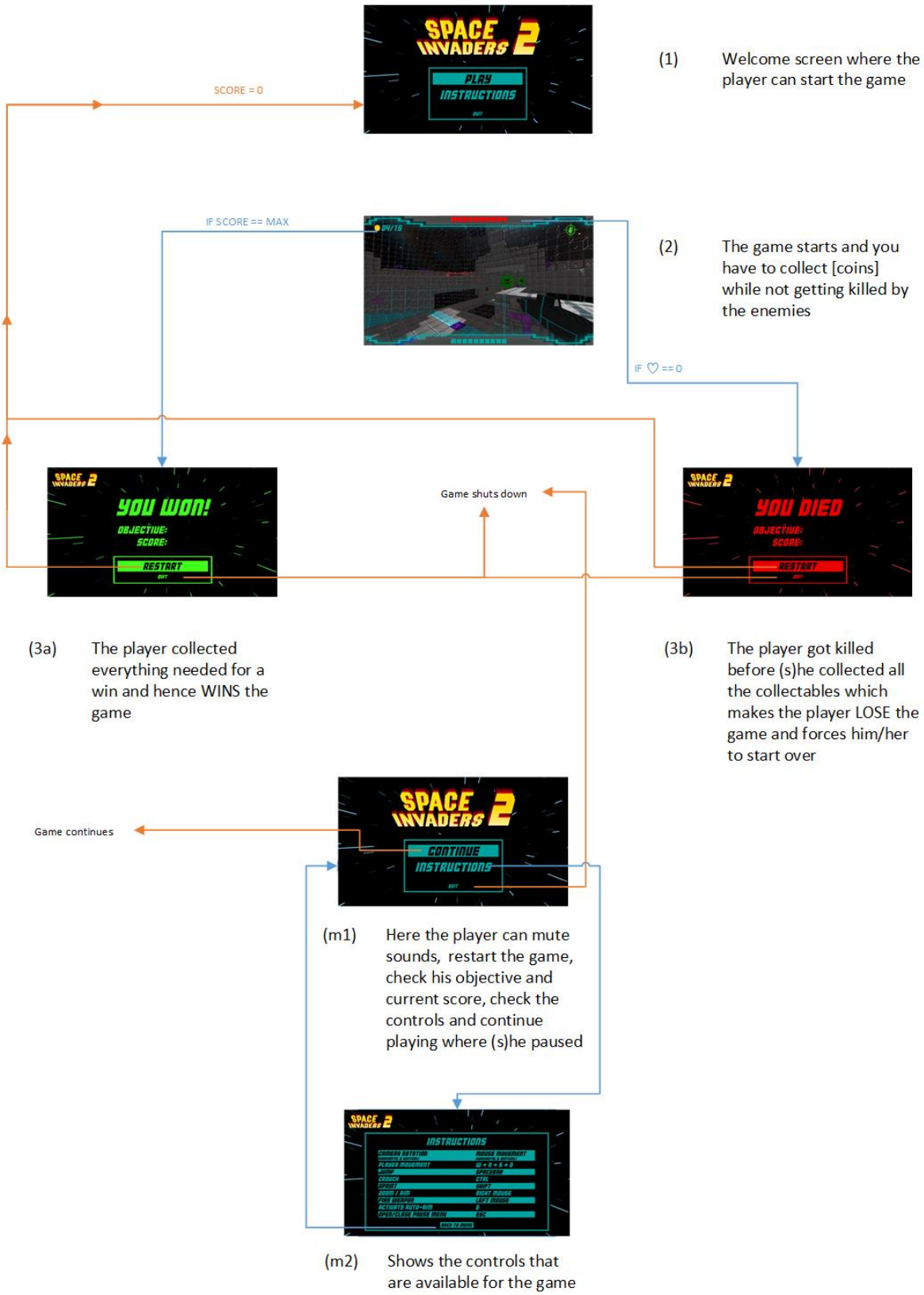

Figure 15: Game structure diagram

B MAP IMAGE

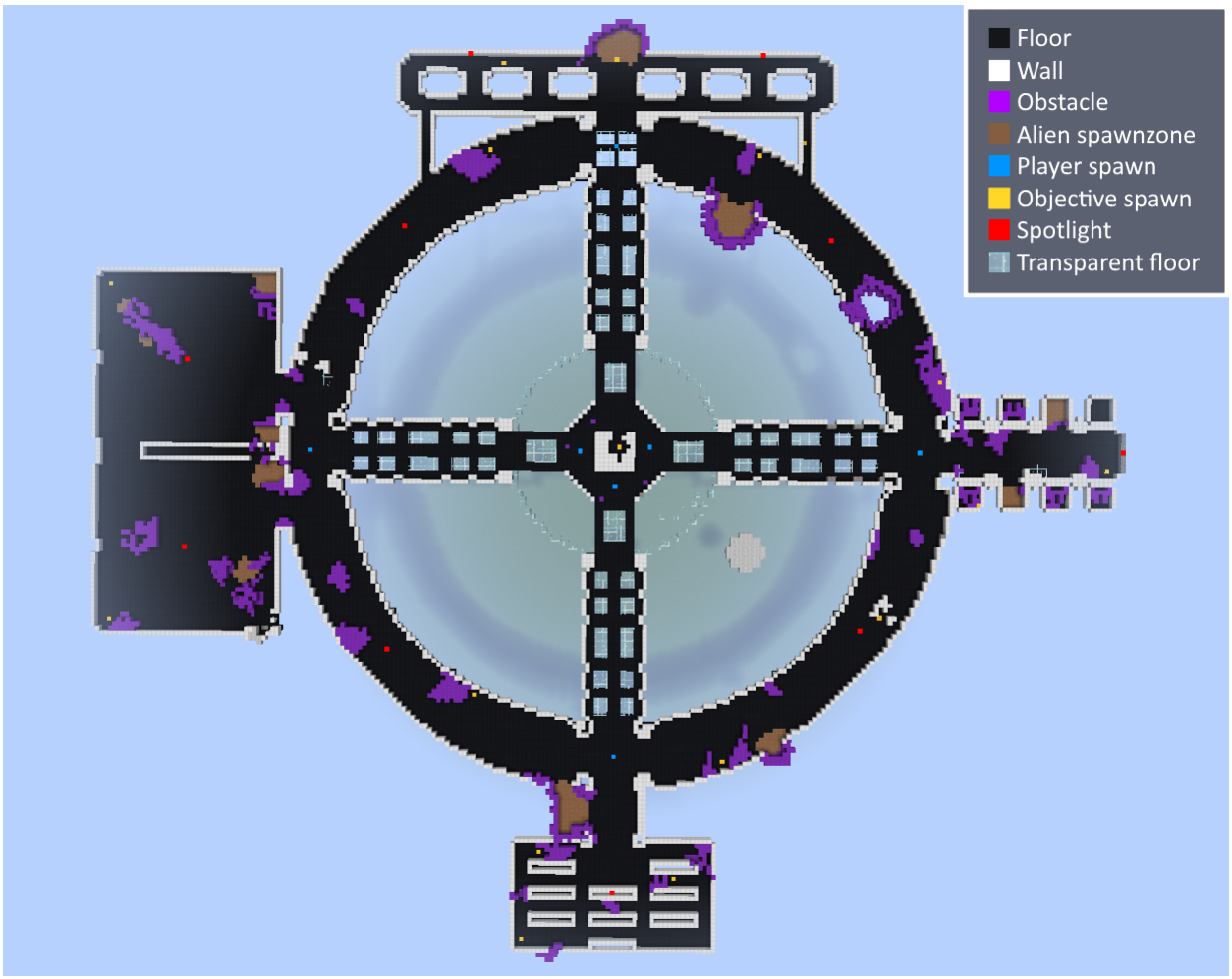

Figure 16: A larger version of the map discussed in Chapter 2.2.

C CONTRIBUTIONS

In this section we state the number of hours that where spend per student. These numbers are specified separately for documentation, realization, presentation, and game design.

	Documentation	**Realization**	**Presentation**	**Game design**	**Total**
Daan Wöltgens	10	74	8	32.5	124.5
Kilian van Berlo	35	68	6	36	145
Mathijs Boezer	11	122	6	31	170
Tobin van den Hurk	21	36	6	43	106
Tom Verberk	10	93	6	32	131

Table 2: Number of hours spend per student

D MEETING NOTES

In this section all the meeting notes will be listed. One note per subsection.

D.1 Meeting notes November 14 2019, 08h45 - 11h15

Minute taker: Kilian van Berlo

Presence:

	Present (Y/N)	Reason (if absent)
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 3: Presence

Number of hours spend per student in the last week:

	Documentation	Realization	Presentation	Game design
Daan Wöltgens	/	/	/	/
Kilian van Berlo	/	/	/	/
Mathijs Boezer	/	/	/	/
Tobin van den Hurk	/	/	/	/
Tom Verberk	/	/	/	/

Table 4: Number of hours spend per student in the last week

To-do list previous meeting:

- /

Major decisions that are taken:

- The game is going to be a first-person shooter where the player has to collect some sort of collectables in order to complete the game.

To-do list for next week:

- Game concept
  - Illustrations (Kilian)
  - Report (Tobin Daan)
- Basis game opstellen (Tom Mathijs)
  - Create and discuss class diagram (Tom)
  - Code basis (Mathijs)
- Read through writing guidelines (Everyone)
- Meeting plannen Huub (Kilian)
  - h.v.d.wetering@tue.nl
  - tobinvandenhurk@gmail.com
  - mathijs.boezer@gmail.com
  - t.m.verberk@student.tue.nl
  - d.p.woltgens@student.tue.nl (daan.woeltgens@gmail.com)

Notes:

- Questions for Huub:
  - Minutes in English or in Dutch?
    English
  - In de studyguide it is noted that a data draft of the minutes should be mailed to the participants and stored in canvas. Stored in canvas how?
    Upload in Canvas group

  - How elaborate should the game concept be?
    See rubric

  - What are interactions?
    How the user interacts with the game:
    * Gesture recognition
    * Selection of parts, obstacles etc.
    * Physics between other objects/NPCs

  - What are conflicts?
    Conflict is what you are, in our case, fighting, about. Most important is that it is clear what it is and how it works, every detail (consistent)!

  - What are animations?
    * Skeleton is an option, no stickman though
    * Morphing
    * Give enough info on techniques to judge whether it is realistic

  - Present concept
    See below for remarks

- Game concepts (2 à 3 pages + illustrations)
  - Something that works is the most important, focus on techniques and not the game on itself per se.
    * CG and AI concepts must serve a (meaningful) purpose within the game.
    * Instead of bad input (not dark) perhaps better to do clear environment when using neural networks.
    * Change the existing model of the world in a voxel world perhaps?
    * Every block is a physical block, or perhaps some of them are not physical.
  - Computer Graphics
    * Advanced shading, for example glass makes it interesting but also harder for surface elimination
    * Visible surface elimination (/interaction)
  - Artificial Intelligence (the challenge is in the training of the learning algorithms):
    * Search: A*, IDA*
    * Neural networks (live image recognition through the sight of the weapon)

- Game Design
  - Players
    * First person
    * One player
    * Adding NPC?
  - Objective
    * Currency collection
    * Survival
    * Story
    * Levels (one or multiple)
  - Procedures
    * Mouse movement
      · Walking
      · Jumping
      · Crouching
      · Looking around
      · Zoom in / out
      · Shooting
    * Menu
      · Escape to pause
      · High scores?
      · Mute button

  - Rules
    - Don't move through objects
    - Don't shoot through objects
    - Cool down the gun (limit on shooting/minute)
    - Player health adjustments (less speed if injured more)
  - Conflicts
    Zombies
  - Outcome
    Complete objective or fail objective

- Theme
  - Sci-fi
- Graphics
  - OpenGL
  - Shading
    - Normal mapping
    - Ray tracing (Shadows, mist, dark/light, water, skybox)
- AI
  - Detection

D.2 Meeting notes November 19 2019, 13h35 - 14h45

Minute taker: Mathijs Boezer

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 5: Presence

To-do list previous meeting:

- Game concept
  - Create and discuss illustrations (Kilian) DONE
  - Report (Tobin Daan) DONE
- Basis game opstellen (Tom Mathijs) DONE
  - Create and discuss class diagram (Tom) DONE
  - Code basis (Mathijs) DONE
- Read through writing guidelines (Everyone) DONE
- Meeting plannen Huub (Kilian) DONE
  - h.v.d.wetering@tue.nl
  - tobinvandenhurk@gmail.com
  - mathijs.boezer@gmail.com
  - t.m.verberk@student.tue.nl
  - d.p.woltgens@student.tue.nl (daan.woeltgens@gmail.com)

Major decisions that are taken:

- The AI will be used for automatically aiming on the enemies instead of just being used for enemy detection
- The player can zoom in using the right mouse and has another way to activate the special power using the AI

- Enemies will avoid walking through each other by stopping when they are about to collide and waiting
- For the map we will have something fairly small with a handful of larger rooms connected by corridors probably in the shape of a space station
- Lights will be voxels too (light emitting block)
- We will use glass to look outside
- No tutorial, but hints and objective and controls in the menu
- Tobin will write about the map design
- Killian will finish the illustrations
- Could keep track of playtime as a score
- Voxels will be 0.5 meters in each dimension

To-do list for next week:

- Tobin will write about the map design and sound design
- Kilian will change the game structure and write about the usage scenario
- Daan will finish the game concept
- Mathijs and Tom will work on the code
- All: Look through the code
- All: Install Intellij and get the project to work on your computer

D.3 Meeting notes November 21 2019, 09h35 - 11h15

Minute taker: Tobin van den Hurk

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Late	Lost his phone

Table 6: Presence

Number of hours spend per student in the last week:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	0	0	0	9
Kilian van Berlo	0	0	0	10
Mathijs Boezer	0	8	0	5
Tobin van den Hurk	0	0	0	8
Tom Verberk	0	4	0	6

Table 7: Number of hours spend per student in the last week

To-do list previous meeting:

- Tobin will write about the map design and sound design MAP DESIGN DONE, SOUND DESIGN IN PROGRESS
- Kilian will change the game structure and write about the usage scenario DONE
- Daan will finish the game concept DONE
- Mathijs and Tom will work on the code DONE
- All: Look through the code DONE
- All: Install Intellij and get the project to work on your computer DONE

Major decisions that are taken:

- We will be using Eclipse (instead of IntelliJ)
- RobotRally library has too many restrictions
- We will be using modern 4.6 openGL instead.
- Peer review, intermediate presentation and intermediate report due Thursday.
- Decide on search algorithm:
  - Dynamic target
  - Avoiding collision
  - Voxel-based
  - Feasible for larger quantities
  - Update as frequently as possible

To-do list for next week:

- Daan, Tom Mathijs: Learn openGL (see notes) (Thursday)
- Tobin Kilian: Make report prepare presentation (Tuesday)
- Next Tuesday we will discuss the report and presentation: Everyone should be able to present.
- For feedback on new version of game design, mail it on Tuesday the latest.

Notes:

- About game concept
  - Was wel goed eigenlijk
  - Illustrations background should be blocks (make clearer)
  - Level design typo: through which (not through with)
  - There are A* algorithms with dynamic targets, alternatively run A* a lot of times.
  - Special power: all enemies visible on the screen or just within a visor?
    - Through visor seems better
    - Multiple target neural network
    - Possible freeze screen upon initializing power
  - Be more concrete in (cg) techniques: Make up our mind!
    - More details
    - Suggestion: (Advanced) global illumination
    - Due to our voxel nature, some specific techniques may be easier for us.
  - Visible surface elimination - what's the difficulty we're addressing? (Z-buffer algorithm)
    - Mostly about our data structure
    - Render scene from light's viewpoint
- Questions
  - What about the presentations and report?
    - Present game concept
    - Possible a demo
    - 2 People randomly have to present
    - About 10 minutes
  - Deadline intermediate report tomorrow?
- Playlist about OpenGL
  https://www.youtube.com/watch?v=VS8wlS9hF8Elist=PLRIWtICgwaX0u7Rf9zkZhLoLuZVfUksDP (OpenGL 3D Game Tutorials from ThinMatrix)

D.4 Meeting notes November 26 2019, 13h30 - 14h45

Minute taker: Tom Verberk
To-do list previous meeting:

- Daan, Tom Mathijs: Learn openGL (see notes) (Thursday)
  Daan and Tom are around tutorial 12, Mathijs is at tutorial 30.
- Tobin Kilian: Make report prepare presentation (Tuesday)
  Tobin Kilian have done some research. They are ready to present the game concept. We have discussed choices:
  - We want to do visible surface elimination, as we don't think that a Z-buffer saves enough processing power.
  - Shading won't be easier with voxels. We will not try to implement voxel cone-tracing, instead we will implement Shadow Mapping.
  - AI techniques:
    * Moving target D* light.
- Next Tuesday we will discuss the report and presentation: Everyone should be able to present. IN PROGRESS
- For feedback on new version of game design, mail it on Tuesday the latest. IN PROGRESS

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	N	Sick
Tom Verberk	Y	/

Table 8: Presence

Major decisions that are taken:

- We are doing visible surface elimination as Z-buffer is not enough.
- We will try to implement Voxel cone tracing.
- Schadow casting is part of advances shading. (Episode 40 of tutorial)
- We are implementing a D* light algorithm.
- Presentation is split into two parts, Split at the size CG technics.
- Tobin won't do the demo part since he is not here now.

To-do list for next week:

- Talk with Tobin about visible surface elimination (Mathijs)
- Tom Update the code diagram (Today)
- Tom, make Open GL library slide for presentation.
- Learn the presentation (both parts)
- Kilian, shadow casting instead of cone illumination.
- Finish intermediate report (Kilian, Tobin)
- If no meeting Thursday, we will discuss TODOs.
- Mathijs, change GUITexture name of texture to textureID

Notes:

- Introduction of game: What are we going to do and with what programs do we work?

D.5 Meeting notes November 28 2019, 09h45 - 10h30

Minute taker: Daan Wöltgens
To-do list previous meeting:

- Talk with Tobin about visible surface elimination (Mathijs) DONE
- Tom Update the code diagram (Today) DONE
- Tom, make Open GL library slide for presentation. DONE
- Learn the presentation (both parts) DONE
- Kilian, shadow casting instead of cone illumination. DONE
- Finish intermediate report (Kilian, Tobin) IN PROGRESS
- If no meeting Thursday, we will discuss TODOs. NOT NEEDED
- Mathijs, change GUITexture name of texture to textureID DONE

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 9: Presence

Number of hours spend per student in the last week:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	0	15	2	5
Kilian van Berlo	17	0	2	5
Mathijs Boezer	0	25	2	5
Tobin van den Hurk	17	0	2	2
Tom Verberk	10	8	2	5

Table 10: Number of hours spend per student in the last week

Major decisions that are taken:

- /

To-do list for next week:

- Hand in intermediate report:
  - Add UML (Tom)
  - Add time spend (Kilian)
  - Add meeting notes (Kilian)
  - Edit CNN (Tobin)
- Tom, Daan: Watch tutorials to 30
- Mathijs: Shadows, Map as file
- Tobin: Neural Network (sentdex GTA)

Notes:

- https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch11.html (Precentage Closer Filtering (PCF))

D.6 Meeting notes December 3 2019, 14h43 - 15h45

Minute taker: Kilian van Berlo

To-do list previous meeting:

- Hand in intermediate report: DONE
  - Add UML (Tom) DONE
  - Add time spend (Kilian) DONE
  - Add meeting notes (Kilian) DONE
  - Edit CNN (Tobin) DONE
- Tom, Daan: Watch tutorials to 30 DONE
- Mathijs: Shadows, Map as file DONE
- Tobin: Neural Network (sentdex GTA)
  Hard to understand, needs help

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 11: Presence

Major decisions that are taken:

- Everyone buys Minecraft (if not yet done already)

To-do list for next week:

- Level design (Tobin)
- MT D*-lite (Tom)
- Enemy design (Daan)
- Texture design (Kilian)
  - Health bar
  - Crosshair
  - Overlay (helmet)
  - Menus
  - Loading screen
- Implement normal mapping (Mathijs)
- Make collisions efficient (Mathijs)

Notes:

- Minecraft voor map design
- Together we selected a texture pack to use

D.7 Meeting notes December 5 2019, 09h10 - 10h45

Minute taker: Mathijs Boezer

To-do list previous meeting:

- Level design (Tobin) IN PROGRESS
- MT D*-lite (Tom) IN PROGRESS
- Enemy design (Daan) IN PROGRESS
- Texture design (Kilian) IN PROGRESS
  - Health bar IN PROGRESS
  - Crosshair IN PROGRESS
  - Overlay (helmet) IN PROGRESS
  - Menus IN PROGRESS
  - Loading screen IN PROGRESS
- Implement normal mapping (Mathijs) IN PROGRESS
- Make collisions efficient (Mathijs) IN PROGRESS

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 12: Presence

Number of hours spend per student in the last week:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	0	10	0	5
Kilian van Berlo	2	3	0	5
Mathijs Boezer	0	15	0	5
Tobin van den Hurk	4	0	0	12
Tom Verberk	0	10	0	5

Table 13: Number of hours spend per student in the last week

Major decisions that are taken:

- Spot lights and possibly shadows later

To-do list for next week:

- Level design (Tobin)
- MT D*-lite (Tom)
- Enemy design (Daan)
- Texture design (Kilian)
  - Health bar
  - Crosshair
  - Overlay (helmet)
  - Menus
  - Loading screen
- Implement normal mapping (Mathijs)

- Make collisions efficient (Mathijs)

Notes:

- Peer review will come soon
- Grades should be fixed soon on canvas
- Shadows explained with the filters applied
- Level created in minecraft with a mod that enables this
- Extra lights can't have shadows on there as well
  - Depends on the light
    - Spots not
  - We have to consider the light sources and how many

D.8 Meeting notes December 10 2019, 13h38 - 14h20

Minute taker: Tobin van den Hurk

To-do list previous meeting:

- Level design (Tobin) IN PROGRESS
- MT D*-lite (Tom) IN PROGRESS
- Enemy design (Daan) IN PROGRESS
- Texture design (Kilian) IN PROGRESS
  - Health bar DONE
  - Crosshair DONE
  - Overlay (helmet) DONE
  - Menus IN PROGRESS
  - Loading screen NOT YET STARTED
- Implement normal mapping (Mathijs) DONE
- Make collisions efficient (Mathijs) DONE

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 14: Presence

Major decisions that are taken:

- Used KD tree, collision checks 50 times faster
  - Only checks collision for closest 100 blocks instead of all blocks
- We need to use threads for animating the loading screen

To-do list for next week:

- Kilian: Texture for pause menu: change play to continue
- Daan & Kilian: Gun texture
- Mathijs: Neural network & loading screen
- Tom & Kilian: Pathfinding
- Tobin: Continue map (including new textures)

Notes:

- Daan made a design for the objective (coin)

D.9 Meeting notes December 12 2019, 09h45 - 10h09

Minute taker: Tom Verberk

To-do list previous meeting:

- Kilian: Texture for pause menu: change play to continue DONE
- Daan & Kilian: Gun texture DONE
- Mathijs: Neural network & loading screen NOT YET STARTED
  Not started on any of these yet, was busy on instruction screen
- Tom & Kilian: Pathfinding IN PROGRESS
  Made a graph of the map. Now looking for implementation
- Tobin: Continue map (including new textures) IN PROGRESS
  Progress has been made, but there were some lighting issues
- Daan: Enemy Design DONE
  Design is done, we want to implement it in the game before Tuesday.

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	N	Got hit by a car last tuesday (sick)
Tom Verberk	Y	/

Table 15: Presence

Number of hours spend per student in the last week:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	0	10	0	5
Kilian van Berlo	0	10	0	5
Mathijs Boezer	0	10	0	5
Tobin van den Hurk	0	0	0	10
Tom Verberk	0	5	0	5

Table 16: Number of hours spend per student in the last week

Major decisions that are taken:

- Daan implements the enemy design in the game before Tuesday
- Not more than 2 lights in an area of 40 blocks. To avoid pop-in of the lights

To-do list for next week:

- Mathijs: Neural network loading screen
- Tom Kilian: Pathfinding, implement A* before next meeting.
- Daan: Implement the design in the game, including animations.
- Tobin: Continue map (including new textures)

Notes:

- Tobin has changed the level, the lighting looks better now.
- Tobin has made too many lights, The level looks not very nice now
- Look at turning lights on and off. For smoother transitions.

D.10 Meeting notes December 17 2019, 13h37 - 14h30

Minute taker: Kilian van Berlo

To-do list previous meeting:

- Mathijs:
  - Neural network IN PROGRESS
  - Loading screen DONE
- Tom Kilian: Pathfinding, implement A* before next meeting. IN PROGRESS
- Daan: Implement the design in the game, including animations. IN PROGRESS
  Encountered some issues with the implementation, he is looking into it at the moment
- Tobin: Continue map (including new textures) DONE

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	N	Sick
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 17: Presence

Major decisions that are taken:

- /

To-do list for next Thursday:

- Tobin: Textures to checkerboard
- Kilian: Write the introduction
- Daan: Design afmaken
  - Animatie implementatie (lopen, slaan, etc.)
  - For the enemy texture: what is now light grey should be bright green
- Mathijs: Neural Network
- Tom + Kilian: Search Algorithm

D.11 Meeting notes December 19 2019, 09h30 - 10h37

Minute taker: Daan Wöltgens

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	N	Got hit by a car last tuesday (sick)
Tom Verberk	Y	/

Table 18: Presence

To-do list previous meeting:

- Tobin: Textures to checkerboard DONE
- Kilian: Write the introduction NOT YET STARTED
- Daan:
  - Animatie implementatie (lopen, slaan, etc.) IN PROGRESS

  - For the enemy texture: what is now light grey should be bright green DONE
- Mathijs: Neural Network IN PROGRESS
- Tom + Kilian: Search Algorithm DONE

Meeting:

- Showed demo

Number of hours spend per student in the last week:

	Documentation	Realization	Presentation	Game design
Daan Wöltgens	0	10	0	2.5
Kilian van Berlo	0	10	0	5
Mathijs Boezer	0	8	0	5
Tobin van den Hurk	0	12	0	5
Tom Verberk	0	11	0	5

Table 19: Number of hours spend per student in the last week

To-do list for next week:

- Tobin:
  - Fix lights
- Kilian
  - Continue report
  - Search algorithm
- Daan:
  - Finish animations
- Mathijs
  - Neural network
- Tom
  - Search algorithm

D.12 Meeting notes January 7 2020, 13h33 - 15h15

Minute taker: Mathijs Boezer

Presence:

	Present (Y/N)	Reason (if absent)
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 20: Presence

To-do list previous meeting:

- Tobin:
  - Fix lights DONE
- Kilian
  - Continue report DONE
  - Search algorithm DONE
- Daan:

  - Finish animations CANCELLED

- Mathijs
  - Neural network DONE
- Tom
  - Search algorithm DONE

Major decisions that are taken:

- MTD* Lite not needed with 50 enemies. A* proves to suffice
- Animations have been cancelled, simpler enemies with morphing instead

To-do list for next week:

- All: playtest game
  - Performance (framerate, smooth movement)
  - Difficulty (surviving, collecting objectives)
  - Bugs/inconsistencies
  - Fun
- Daan:
  - Make presentation
- Tobin:
  - Make demo video
  - Fix are under UFOs
- Mathijs:
  - Variable enemy speed
- Tom:
  - Fix search algo (diagonal movement)
  - Fix enemies going through walls and similar bugs
- Report:
  - Mathijs: Neural network, CG topics
  - Kilian: Search algorithm

Questions for Huub:

- Report: Explain changes in our approach (MTD* Lite to A*) OR only explain our final decision? Research paper without research?
- Validation tools: How explicitly?
- Demo: Standalone demo (presenters give commentary) or built-in commentary (via text or audio)?
- Question in agenda for 9/1

D.13 Meeting notes January 9 2020, 09h45 - 11h30

Minute taker: Tom Verberk

To-do list previous meeting:

- All: playtest game DONE
  - Performance (framerate, smooth movement) DONE
  - Difficulty (surviving, collecting objectives) DONE Collecting all the coins if they respawn seems boring
  - Bugs/inconsistencies DONE
  - Fun DONE
- Daan:
  - Make presentation IN PROGRESS
- Tobin:
  - Make demo video NOT YET STARTED
  - Fix are under UFOs DONE
- Mathijs:
  - Variable enemy speed DONE
- Tom:
  - Fix search algo (diagonal movement) DONE
  - Fix enemies going through walls and similar bugs DONE
- Report:
  - Mathijs: Neural network, CG topics NOT DONE
  - Kilian: Search algorithm NOT YET STARTED

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	Late
Kilian van Berlo	Y	Late
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	Late
Tom Verberk	Y	/

Table 21: Presence

Number of hours spend per student in the last week:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	0	20	2	5
Kilian van Berlo	3	35	0	5
Mathijs Boezer	0	40	0	5
Tobin van den Hurk	0	10	0	5
Tom Verberk	0	35	0	5

Table 22: Number of hours spend per student in the last week

Major decisions that are taken:

- Implement increasing difficulty
- On/off shadow rendering

Demo:

- Brief introduction, show map
- First explain techniques

- Morphing, Shadow Mapping, neural network, bump mapping, collision, spotlights
- GUI explanation, show controls/menus
- Tutorial on how to play
- General gameplay

To-do list for next week:

- Daan:
  - Presentation
  - Take a microphone with you next tuesday
- Tobin:
  - Demo
  - Out of map bug
- Mathijs:
  - Increasing difficulty
  - On/off Shadow Mapping
- Tom:
  - Fix bugs in A*
- Report:
  - Mathijs: Neural network, CG topics
  - Kilian: Search algorithm
  - Daan + Tobin: Intro, abstract, conclusion

Notes:

- Use Output tensorflow as verification of the Neural network. If you use this as proof together with some screenshots from the game you will come far.
- For A* as verification it is enough to make a testworld.
- For presenting Morphing in the presentation we can give an enemy 0 speed and make a movie. In the report we should explain something like the 4 different stages.
- For Shadow Mapping we could make a button that turns it on and off for the demo. This way people will see the difference. In the report we need to explain how we do it.
- Video for the demo should have a minimum of 3 minutes.
- The deadline of the report is the 19th.
- Explain in the report why we first tried MTD* but stayed at A*.
- Intro should be relatively short. The ideas of the game could be in here.
- Presentation will be around the same as last time. During presentation video.
- Handing in the source code, compile and in one file.

D.14 Meeting notes January 14 2020, 13h43 - 15h30

Minute taker: Daan Wöltgens

To-do list previous meeting:

- Daan:
  - Presentation DONE
  - Take a microphone with you next tuesday DONE
- Tobin:
  - Demo DONE
  - Out of map bug DONE
- Mathijs:
  - Increasing difficulty DONE
  - On/off Shadow Mapping DONE
- Tom:
  - Fix bugs in A* DONE
- Report:
  - Mathijs: Neural network, CG topics DONE
  - Kilian: Search algorithm NOT DONE
  - Daan + Tobin: Intro, abstract, conclusion NOT DONE

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 23: Presence

To-do list for next week:

- Fix intro (name was wrong)
- Add sprint to menu
- Reorganize Menu
- New Shadow picture for presentation
- New video intro

Presentation split:

- 1-6
- 7-10
- 11-13
- 14-End

D.15 Meeting notes January 16 2020, 09h45 - 11h30

Minute taker: Kilian van Berlo

To-do list previous meeting:

- Fix intro (name was wrong) DONE
- Add sprint to menu DONE
- Reorganize Menu DONE
- New Shadow picture for presentation DONE
- New video intro DONE

Presence:

	Present (Y/N)	**Reason (if absent)**
Daan Wöltgens	Y	/
Kilian van Berlo	Y	/
Mathijs Boezer	Y	/
Tobin van den Hurk	Y	/
Tom Verberk	Y	/

Table 24: Presence

Number of hours spend per student till final submission:

	Documentation	**Realization**	**Presentation**	**Game design**
Daan Wöltgens	10	8	4	1
Kilian van Berlo	13	10	4	1
Mathijs Boezer	11	16	4	1
Tobin van den Hurk	9	14	4	1
Tom Verberk	0	20	4	1

Table 25: Number of hours spend per student in the last week

To-do list for submission:

- All: Read through report
- Kilian: Minutes, references, appendix
- Kilian Tom: AStar
- Tom: Extra features (bewegende skybox, spotlights, bump/normal mapping, collision detection)
- Tom Daan: Adjust intermediate report part
- Daan Tobin: Intro, abstract conclusion
- Tobin: Peer review
- Mathijs: Performance
- Mathijs: Finish game