# Alpha Report

## 1. Low Target

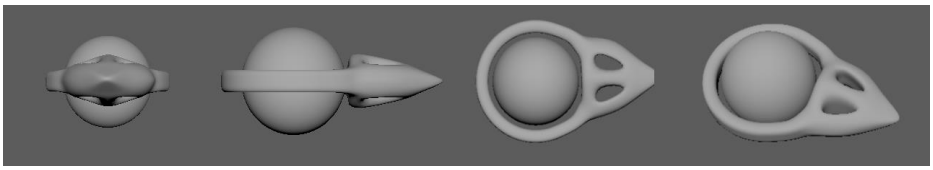
### Weapon Heat

To ensure a certain amount of strategy and avoid overloading the arena with projectiles, we needed a game mechanic to prevent the player from continuous firing. Since a bullet capacity/reloading system doesn't really make sense in our light approach, we've decided to implement a weapon overheat system: Each time the player fires a projectile, his weapon overheat increases. Once it reaches the limit the weapon is overheated, the player isn't able to keep shooting and has to wait until his weapon has cooled down. Additionally, the higher the overheat of a weapon is, the lower energy the projectile it fires has. To balance out the drawbacks of low energy projectiles, they will produce smaller overheat than high energy projectiles. So the weapon overheat doesn't increase in a linear manner, but somewhat logarithmically. It looks as follows:

| Current Weapon Overheat | Increase |
|---|---|
| 0% - 5% | +30% |
| 5% - 15% | +25% |
| 15% - 30% | +20% |
| 30% - 60% | +15% |
| 60% - 100% | +10% |

### Own Player Model

We replaced the player model, which was given by Unreal, by a basic, but own 3D model. The model was inspired by rolling robots, which consists of a spherical body. The spherical body ensures the desired movement in arbitrary direction. An additional ring around the body represents the gun. This gun forms an arrow at one side to clearly indicate the shooting direction. The gun ring is detached from the body and simple floats in the air. This again ensures the 360° shooting direction.



### Advanced Projectiles: Colors

The difficulty in applying the different colors regarding the projectile's energy level was to find an appropriate transition function between the colors blue over green to red. This was achieved by using the HSV colors, where the saturation and the value (brightness) both were kept at 1.0f, whereas the hue ranged from 240 for blue to 0 for red (120 is green), depending on the current energy level. This HSV color was then transformed into linear RGB values to change the final color. The Unreal Engine dependent property was to reapply the material to the static mesh to see the change in color, as UE can't have runtime changeable materials directly.

**Advanced Obstacles: Multiple materials**

To increase the variety of obstacles for enhancing strategic gameplay, multiple obstacle types were added with different refraction and absorption coefficients. The coefficients are based on the materials gold, glas, silicon, germanium and crystal/diamond. The decision for those types are based on the availability of the refraction and absorption coefficients, which are e.g. not really defined for organic materials such as wood, but very well investigated for inorganic materials such as different types of plastic and metals. Another criterium is the visible distinction between the materials, so the player is able to determine the different obstacle types easily.

The types are implemented using inheritance. Therefore, a basic voxel class is created and then the specific material coefficients and textures are defined in the child classes.

## 2. Desirable Target

**Obstacle destruction: Coupling with projectiles**

The obstacle destruction is managed by a global destruction handler. Since there is no master class which has an overview over all actors in the game, the destruction handler has to be stored in the projectiles, which are calling the test function after they damaged an obstacle. To ensure that not every projectile has its own destruction handler, but there is one global for every projectile in the scene, a singleton container is used. Every time a projectile is shot, the construtor checks if already a global destruction handler is available and if yes, sets the pointer within the projectile class. However, the first time a projectile is created, no global destruction handler is available, so that one will be created.

**Advanced Reflections**

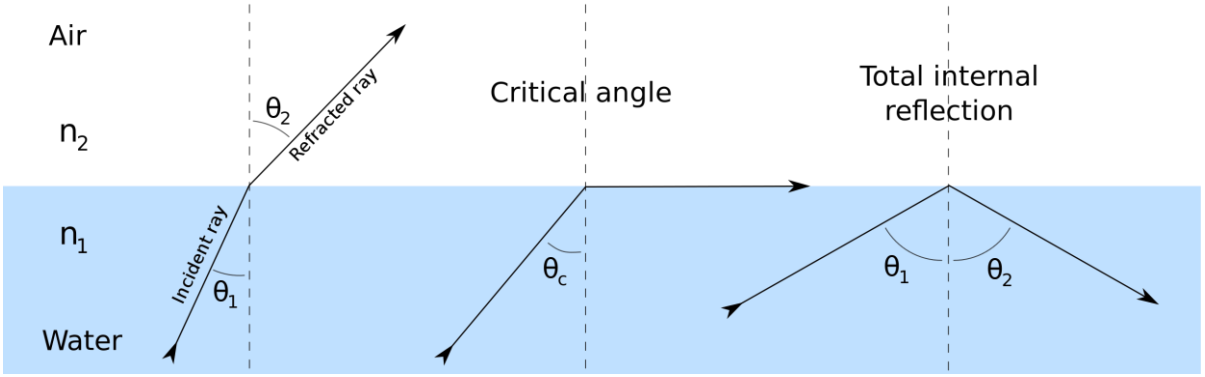
For advanced reflections, we needed to implement physical properties of light-matter-interactions. In reality, these can become quite difficult for certain materials, as they use the polarity of light to get the final refraction and reflection directions and intensities. For our real time application, we needed something less detailed and more focused on fast results. Therefore to receive the reflection coefficient (amount of reflection in contrast to refraction), we used Schlick's approximation for the contribution of the Fresnel factor in the specular reflection of light. This computes the relation of the reflection coefficient parallel to the normal the amount of difference with the cosine between incident light and normal. For this calculation, we also need refractive indices for the two materials, one being the air and set to 1.0f, the other one depending on the hit material.

Also for the light being refracted into the material, we need to change the direction. This is done by applying Snell’s Law with the formular:

$$n_1 * \sin\theta_1 = n_2 * \sin\theta_2$$

It changes the direction on the interface between two materials regarding their refractive indices and the incoming light direction. This is done for light entering and leaving a material. Therefore we need the care for sine values bigger than 1.0f, as these will result in numerical

errors if not handled. The reason for this occurrence is the total internal reflection.




**Advanced Projectiles: Light trail**

As light traverses pretty fast through the world, we cannot just use a simple capsule for a photon as the players would never be able to see them.

Because of the speed of the projectiles, we use a particle system component for gameplay and visibility reasons. This is done by the Unreal Engine's Ribbon Type Data.

This emits some particles when following the projectile and connecting them, rendering the trail with some triangles. The result is a visible trail following the light projectile so that players can predict the direction and possibly evade the damage.

**Menu: Main, Resolution, Winning, Pause, Join/Start**

The basic menu has been extended to ensure a complete gameflow. The main menu offers now a resolution option for 640x480, 1280x720 and 1920x1080. Furthermore, a controller help view is added to explain the controller handlings.

In the arena level, the players have to join the game to ensure, the correct number of active players are added. Once, all players joined, the first player controller can start the game. During the gameplay, a pause menu can be called, which offers the option to quit the game and return to the main menu. While the pause menu is open, the game itself is set to pause mode.

After all players are killed except of one, a winning screen appears with the option to restart the game or return to the main menu. If the game is restarted, the players need to join again. This makes sure, new players can join as well without the need to go back to the main menu.

**Random Map Generator: Use multiple obstacle types**

In conjunction with the multiple obstacle types the Random Map generator has been extended. It now randomly chooses the type of the obstacle with equal probability distribution.

## 3. High Target

**Visual improvements: Menu, HUD**

With the intention to make our HUD more appealing and fit better to our game setting, we need to add some visual improvements to make it look more futuristic. Therefore, we decided to use a science-fiction looking font. Furthermore, we added some icons to indicate the health and overheat bars.

Additionally, all menus have been redesigned. The titles of the menus now use a rainbow-colored gradient, representing the color spectrum of the projectiles. The design of the buttons also follows the futuristic setting.

The overall interface is now much more appealing and also follows a clearer design.

**Player distance zoom**

The prior static virtual camera was replaced with a dynamic one. This dynamic camera now ajdusts its position according to the position of the players. Also it zooms in, if the players move closer to each other and zooms out, if they separate, respectively.

Since it is important to have a minimum knownledge about the surrounding arena, the maximum zoom-in level is specified.

**Smooth Player rotation**

Currently if the player decides to change the direction of his character, the rotation happens instantaneously. To smoothen the movement, the rotation of the character has to interpolate between its current angle and the goal angle.

**Detailed Reflections**

If we want to support more realistic reflections and refractions, we need to consider our energy level of the light projectile and its depending wave length. With this in mind, we can have more detailed effects, so we have added for the different materials additional fix points for wave lengths between 400 µm (blue) and 700 µm (red), in number 7 points, so values in between will get interpolated linearly to get the respective refractive indices and extinction coefficients for absorption and damage application. Our information is taken from the site http://refractiveindex.info .

## 4. Design revisions

While most game elements currently only provide alpha-status functionality, we did not change any aspect of our initial design decisions.

## 5. Challenges

Again as before, we have been faced with some counter-intuitive behaviours and restrictions of the Unreal Engine.