Graphics Report

Contents

Sound

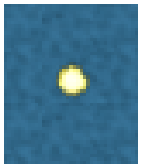
Ambient water sounds

```
PlaySound("sounds/water.wav", NULL, SND_ASYNC | SND_FILENAME | SND_LOOP);
```

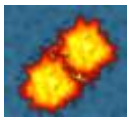
Animations

Explosion Animation

Shot



Enemy Death



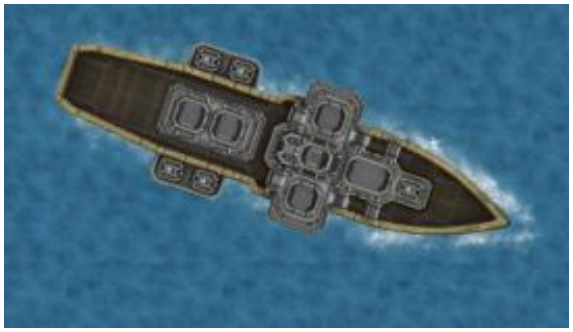
Explosion Pseudo Code

In Explosion object render function:
(hp starts at 18(*number of frames in animation*))

```
glBindTexture(GL_TEXTURE_2D, explosion[i]);
i++;
hp--;

if(hp <= 0){
        Delete This object;
}
```

Player Movement animations

Forward & Backwards



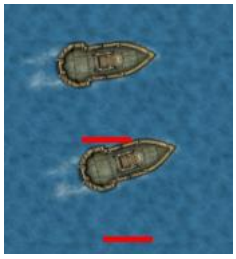
Player Animation Code

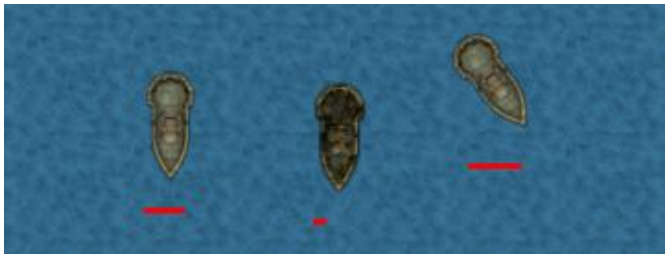
texture = no wake.
animation[0-4] = moving forward.
animaton[5-9] = moving backwards.

```
if (Vtri < -0.01){
        glBindTexture(GL_TEXTURE_2D, animation[i + 5]);
}
```

```
else if (Vtri > 0.01){
        glBindTexture(GL_TEXTURE_2D, animation[i]);
}
else{
        glBindTexture(GL_TEXTURE_2D, texture);
}
i++;
```

Enemy movement, damaged texture & Health Bars





Enemy Damage and movement animations Code

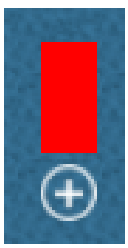
```
if (hp>0.3){
    if (Vtri < -0.01){
        glBindTexture(GL_TEXTURE_2D, animation[i + 6]);
    }
    else if (Vtri > 0.01){
        glBindTexture(GL_TEXTURE_2D, animation[i + 1]);
    }
    else{
        glBindTexture(GL_TEXTURE_2D, animation[0]);
    }
}
else{
    if (Vtri < -0.01){
        glBindTexture(GL_TEXTURE_2D, damage[i + 6]);
    }
    else if (Vtri > 0.01){
        glBindTexture(GL_TEXTURE_2D, damage[i + 1]);
    }
    else{
        glBindTexture(GL_TEXTURE_2D, damage[0]);
    }
}
```

Enemy health Bar Code

```
//Draw Enemy Health Bar
glPushMatrix();
glLoadIdentity();
glTranslatef(Xtri-2.5, Ytri-8, 0.0);
    glBegin(GL_QUADS);
    glColor3f(1, 0, 0);
    glVertex2f(0, 0);
    glVertex2f(hp * 5, 0);
    glVertex2f(hp * 5, 0.5);
    glVertex2f(0, 0.5);
    glEnd();
glPopMatrix();
```

Player Health And Ultimate Weapon cooldown Bars

drawHealth() & drawCooldown() functions used to draw both bars.



Player Health and Cooldown bar Code

```
glBegin(GL_QUADS);
glColor3f(1, 0, 0);
glVertex2f(0, 0);
glVertex2f(5, 0);
glVertex2f(5, player->hp * 10);
glVertex2f(0, player->hp * 10);
glEnd();
```

```
if (ultiCooldown < ultiCooldownMax){
    glColor3f(.112, .138, .144);
}
else{
    glColor3f(0, 1, 0);
}
glVertex2f(0, 0);
glVertex2f(5, 0);
glVertex2f(5, ultiCooldown / 10);
glVertex2f(0, ultiCooldown / 10);
glEnd();
```

Upgrades

Heal

```
//Heal
if (keys['Q'])
{
    if (scrapLevel >= upgradeCost && objects[playerID]->hp < 1 && keyCount <= 0){
        objects[playerID]->hp = 1;
        scrapLevel -= upgradeCost;
        keyCount = maxKeyCount;
    }
}
```

Number of shots

Min Upgrade

Max Upgrade

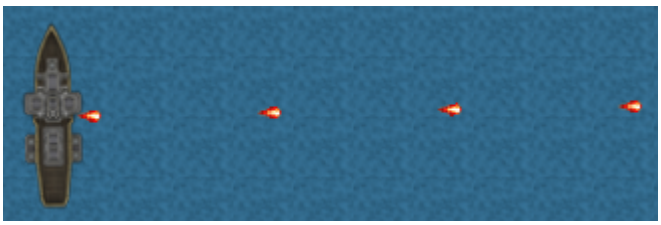
Code

```
//Upgrade Num Shots
if (keys['4'])
{
    if (scrapLevel >= upgradeCost && keyCount <= 0){
        if (shotCount<upgradesMax){
            scrapLevel -= upgradeCost;
            keyCount = maxKeyCount;
            shotCount++;
        }
    }
}
```

Fire rate

Min Upgrade

Max Upgrade

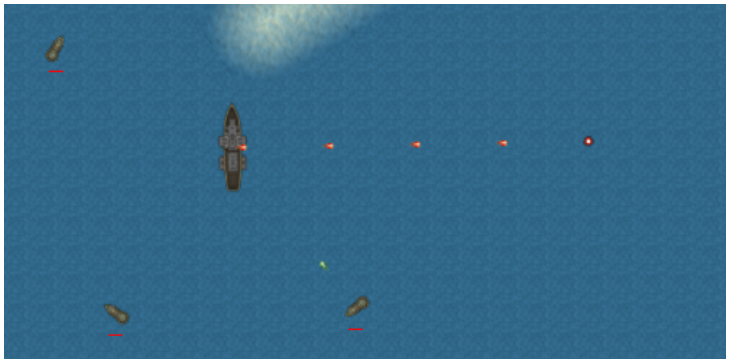


Code

```
//Upgrade Fire Rate
if (keys['3'])
{
    if (scrapLevel >= upgradeCost && keyCount <= 0){
        if(fireRateCount<upgradesMax){
            cooldownReset -= 8;
            scrapLevel -= upgradeCost;
            keyCount = maxKeyCount;
            fireRateCount++;
        }
    }
}
```

Range

Min Upgrade



Max Upgrade

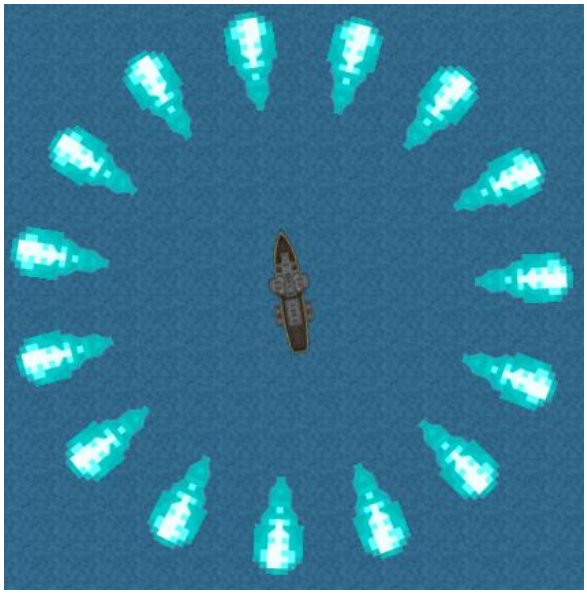
Code

```
//Upgrade Shot Range
if (keys['2'])
{
    if (scrapLevel >= upgradeCost && keyCount <= 0){
        if(rangeCount<upgradesMax){
            range += 0.5;
            scrapLevel -= upgradeCost;
            keyCount = maxKeyCount;
            rangeCount++;
        }
    }
}
```

Speed

```
//Upgrade speed
if (keys['1'])
{
    if (scrapLevel >= upgradeCost && keyCount <= 0){
        if(speedCount<upgradesMax){
            maxSpeed += 0.25;
            scrapLevel -= upgradeCost;
            keyCount = maxKeyCount;
            speedCount++;
        }
    }
}
```

Ultimate Weapon



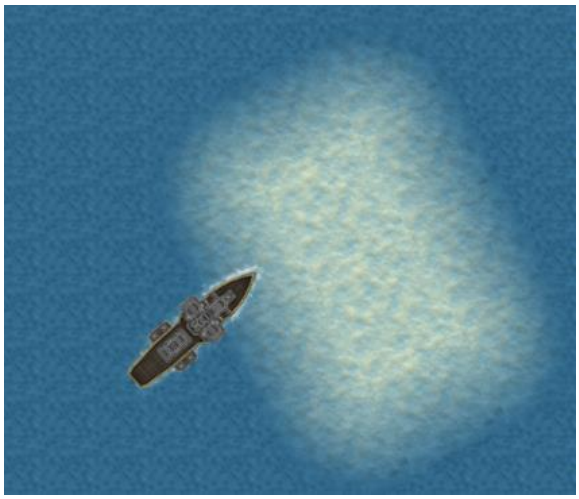
Code Snippets

```
if (keys[VK_SPACE]){
    if (ultiCooldown >= ultiCooldownMax && keyCount <= 0){
        objects.push_back(new Shot(2, player->heading      , player->Xtri, player->Ytri, 3,
          ultiShot, "ultiShot", 6));
        objects.push_back(new Shot(2, player->heading + 24 , player->Xtri, player->Ytri, 3,
          ultiShot, "ultiShot", 6));
        objects.push_back(new Shot(2, player->heading + 48 , player->Xtri, player->Ytri, 3,
        objects.push_back(new Shot(2, player->heading + 336, player->Xtri, player->Ytri, 3,
          ultiShot, "ultiShot", 6));

        keyCount = maxKeyCount;
        ultiCooldown = 0;
    }
```

Homogenous Terrain

Shallow Water

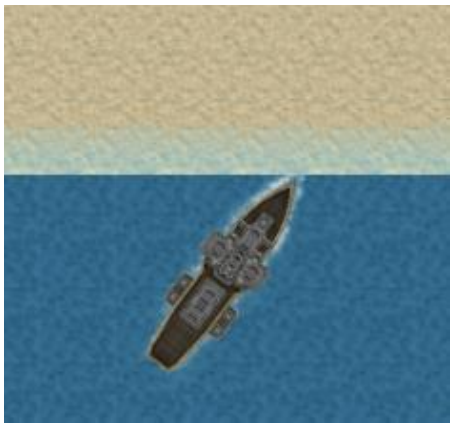
Slows the player when sailing over it.



```
void Player::CollisionReaction(BaseObject* obj){
    if (obj->getType().compare("shallows") == 0){
        //Slow down
        if (Vtri > 1){
            Vtri = 0.5f;
        }
        else if (Vtri > 0.1f){
            Vtri = 0.05f;
        }

    }
```

Beach

Damages the player on collision and reverses movement so is impassable.



Enemy collide

When the player collides with an enemy it damages both the player and the enemy and the player will bounce off.

In players collisionReaction():

```
if (obj->getType().compare("enemy") == 0){
    hp -= 0.001f;
    Vtri = -0.3f;
}
```

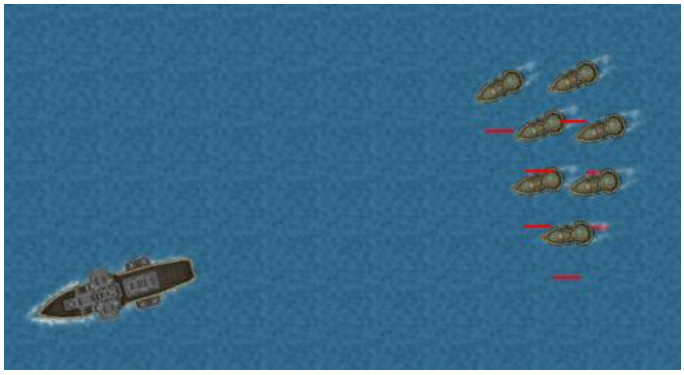
In enemies collisionReaction():

```
//Enemy collide with player
if (obj->getType().compare("player") == 0){
    hp -= 0.025f;
}
```

Enemy AI

Movement

When you get within a certain range of an enemy they will begin to move towards you. This is done by drawing a line between the enemy and the player then performing actions based on its length.



```
dir.x = playerX - enemy->Xtri;
dir.y = playerY - enemy->Ytri;
hyp = sqrt(dir.x*dir.x + dir.y*dir.y);
```

```
float enemyHeading = std::atan2(-dir.x, dir.y);
enemyHeading = (enemyHeading *(180.0f / PI));
if (enemyHeading<0){
    enemyHeading += 360;
}
```

```
//Move to player
if ((hyp < enemydetectRange && hyp>enemyShootRange + 5) ||
    (enemy->spotted && hyp>enemyShootRange+5)){
    enemy->spotted = true;
    enemy->heading = enemyHeading;
    enemy->Vtri = 0.5f;
}
```

Shooting

When they get close enough they will turn broadside and begin to fire at you.

```
//Shoot in range
if ((hyp < enemyShootRange && hyp>10)){
    enemy->heading = enemyHeading + 90;
    enemyShoot(enemy);
}
```

Collision

Enemies correctly collide with each other so they do not overlap as seen in the image above.

```
void enemyAvoid(BaseObject* obj1, BaseObject* obj2){
    Point dir;
    float hyp;

    dir.x = obj2->Xtri - obj1->Xtri;
    dir.y = obj2->Ytri - obj1->Ytri;

    hyp = sqrt(dir.x*dir.x + dir.y*dir.y);
    dir.x /= hyp;
    dir.y /= hyp;

    if (hyp < 10 && hyp>0){
        obj1->Xtri -= dir.x*0.5;
        obj1->Ytri -= dir.y*0.5;
        obj2->Xtri += dir.x*0.5;
        obj2->Ytri += dir.y*0.5;
    }
}
```

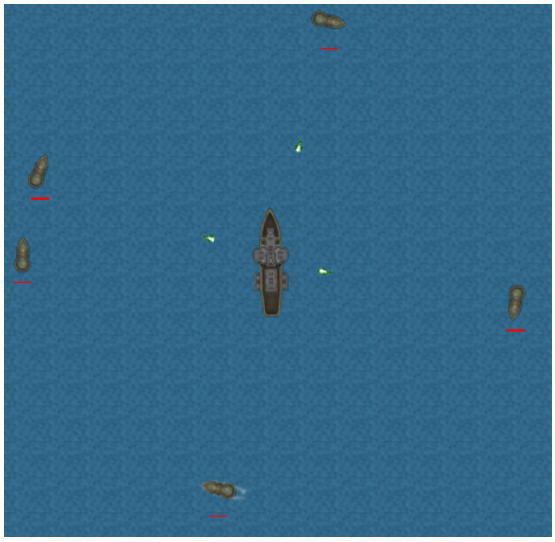
When an enemy collides with the edge of the map or the boss they will change their heading based on the angle they collided at.

```
//Enemy collide with boarder or boss
else if (count <= 0 && (obj->getType().compare("boarder") == 0 ||
    obj->getType().compare("boss") == 0)){
    heading += 180;
    if (heading>360){
        heading -= 360;
    }
    count = 5;
}
```

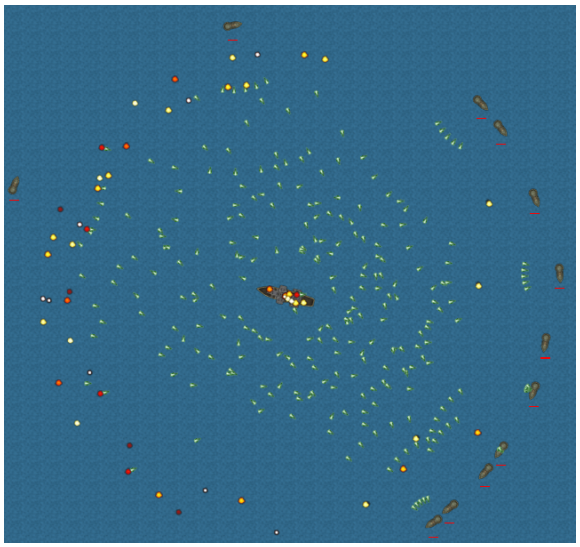
Difficulty progression

As you kill more enemies they will become harder, detecting you from further away, firing more shots and faster and shooting from further away.

Standard enemies



Fully upgraded enemies



Pseudo Code

Shot creation code:
new Shot(enemyRange, enemy->heading - 70, enemy->Xtri, enemy->Ytri, 3, enemyShotTexture, "enemyShot")

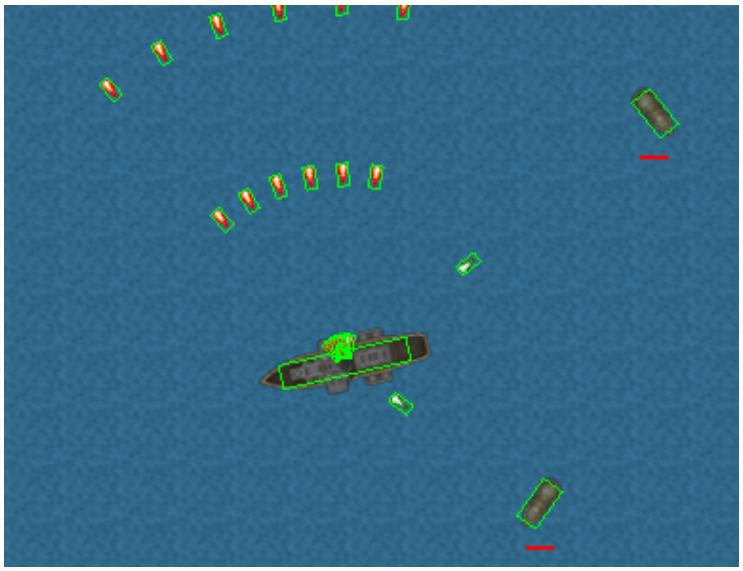
in enemyShoot Function:
If(Number of kills > 40){
        Create 5 shot objects;
}
else If(Number of kills > 30){
        Create 4 shot objects;
}
else If(Number of kills > 20){
        Create 3 shot objects;
}
else If(Number of kills > 10){
        Create 2 shot objects;
}
else If(Number of kills > 0){
        Create 1 shot object;
}

OBB Collisions

All collision are done using Oriented bounding boxes. Only certain objects need to collision check with each other so the shouldCollide() function in main.cpp is used to determine what should collide with what. E.g. –

```
if (obj2type == "player" && obj1type == "enemy"){
    return true;
}
if (obj1type == "enemy" && obj2type == "playerShot"){
    return true;
}
```

Collision Boxes Drawn



UI Text

All text is drawn using the freeType library and its print method. E.g.

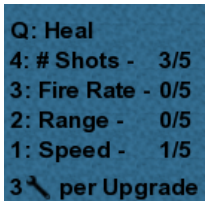
```
print(font, screenWidth - 190, screenHeight - 39, "%d/%d", kills, reqKills);
```

Kills & Scrap Level

drawKills() & drawScrap() functions

Upgrade information

drawUpgrades() function



Boss Spawned

When the player has reached the required amount off kills the boss will spawn and this message will display -

Low Health

When the players health is low a warning appears -

Boss AI

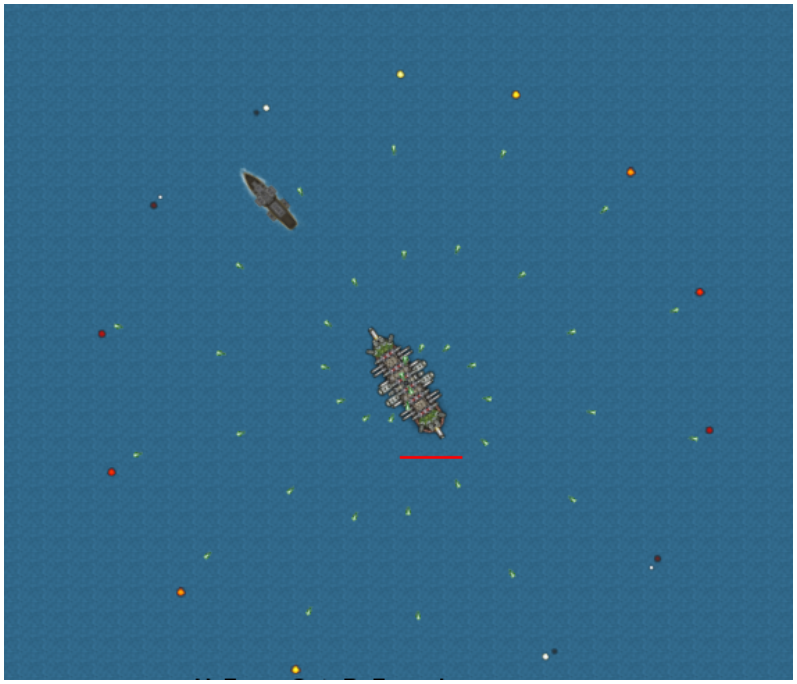
Movement

When the boss has spawned it will move towards the player from any distance. Similar to the standard enemies the movement is done by drawing a line between the player and the boss and the boss will change its heading to follow the line and move along it.

```
dir.x = playerX - boss->Xtri;
dir.y = playerY - boss->Ytri;
hyp = sqrt(dir.x*dir.x + dir.y*dir.y);

float enemyHeading = std::atan2(-dir.x, dir.y);
enemyHeading = (enemyHeading *(180.0f / PI));
if (enemyHeading<0){
    enemyHeading += 360;
}

if ((hyp>100)){
    boss->heading = enemyHeading;
    boss->Vtri = 1;
}
```

Shooting

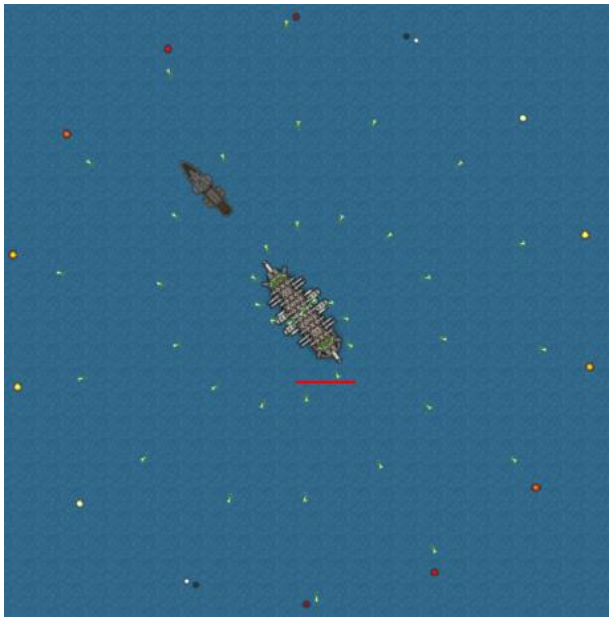
Unlike the normal enemies, the boss will shoot in a spiral around itself, rather than just shooting directly at the player.



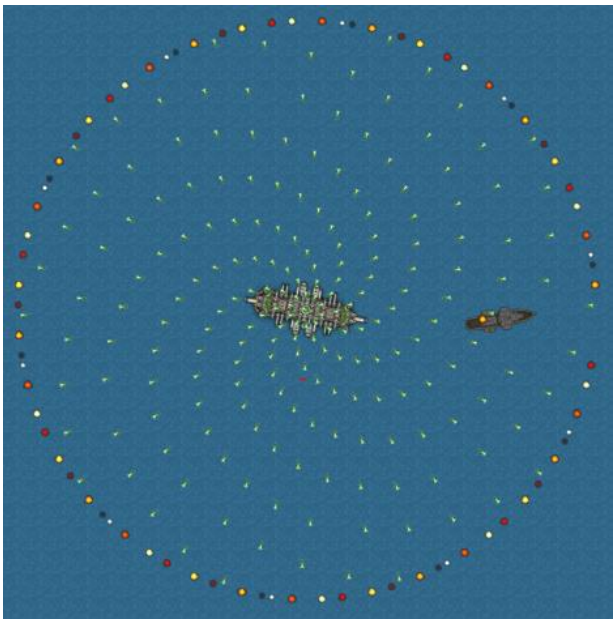
Difficulty Progression

As the bosses health lowers, the amount of shots that it fires will increase. This is performed in the moveBoss() & bossShootCircle() functions in main.cpp.

Standard shooting



Maximum Shooting



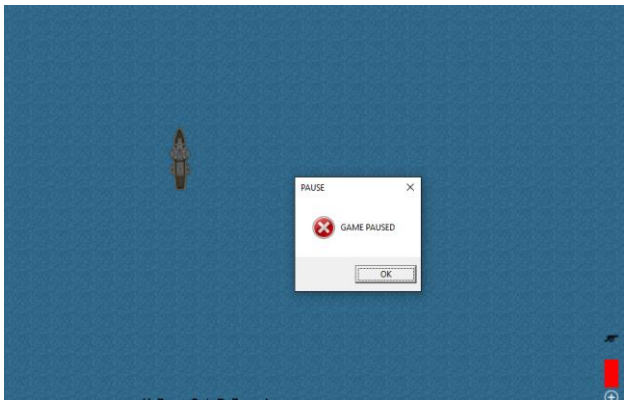
Scrap

When the player kills an enemy a scrap object will drop, and if it within a certain range of the player it will move towards them.

```
void moveScraptoPlayer(BaseObject* scrap){
    Point dir;
    float hyp;

    dir.x = playerX - scrap->origX;
    dir.y = playerY - scrap->origY;
    hyp = sqrt(dir.x*dir.x + dir.y*dir.y);

    float scrapHeading = std::atan2(-dir.x, dir.y);
    scrapHeading = (scrapHeading *(180.0f / PI));
    if (scrapHeading<0){
        scrapHeading += 360;
    }

    //Move to player
    if ((hyp < 200 && hyp>0)){
        scrap->heading = scrapHeading;
        scrap->Vtri = maxSpeed +1;
    }
}
```

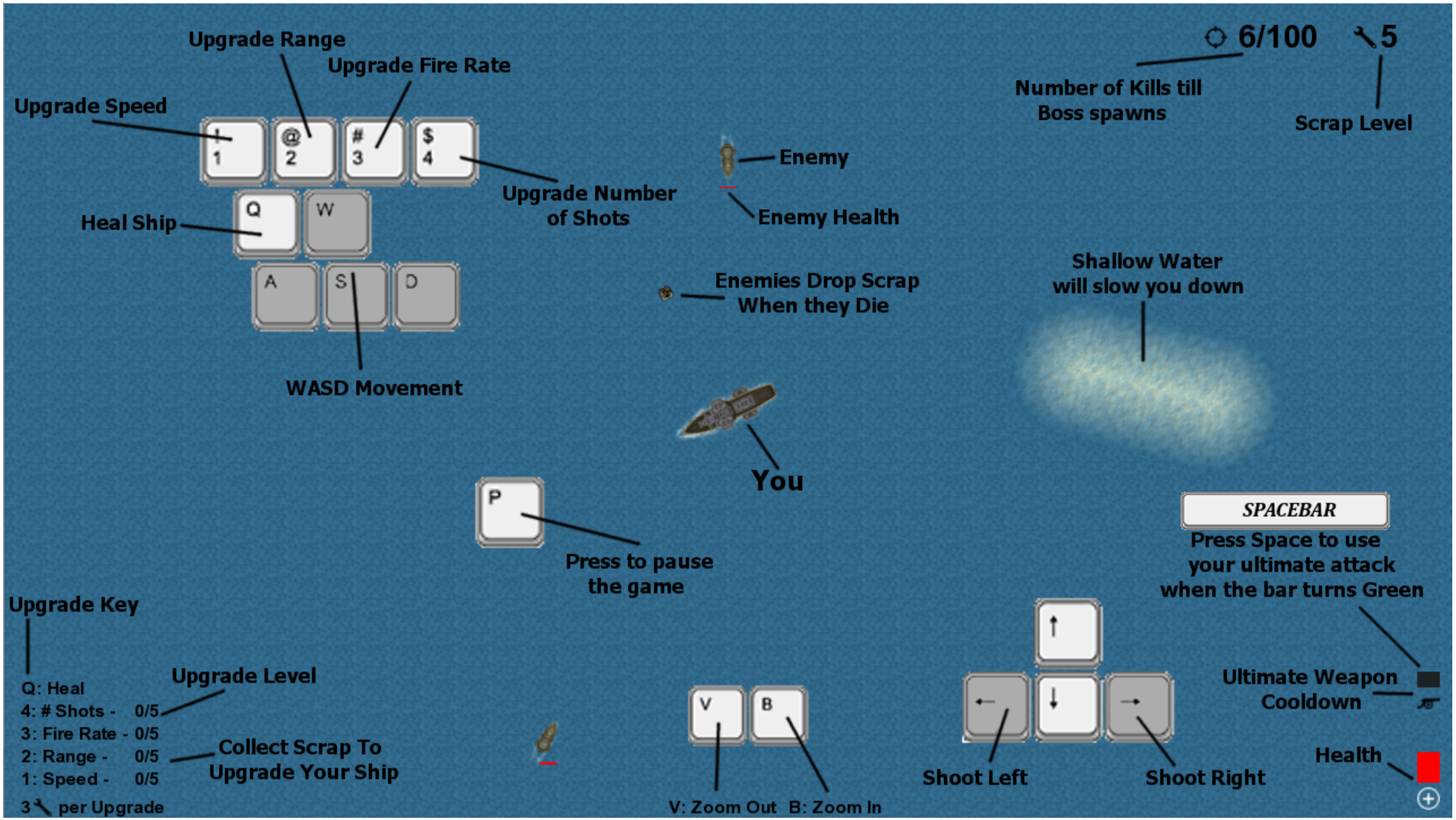
Pause Game

Pressing the ‘P’ key will pause the game and show a message prompt.




```
if (keys['P'] ){
    if (keyCount <= 0){
        paused = MessageBox(NULL, "GAME PAUSED", "PAUSE", MB_OK | MB_ICONSTOP);
        keyCount = maxKeyCount;
        keys['P'] = false;
        keys['V'] = false;
        keys['B'] = false;
        keys['1'] = false;
        keys['2'] = false;
        keys['3'] = false;
        keys['4'] = false;
        keys['W'] = false;
        keys['A'] = false;
        keys['S'] = false;
        keys['D'] = false;
        keys['H'] = false;
        keys[VK_LEFT] = false;
        keys[VK_RIGHT] = false;
    }
}
keyCount--;
```

User Guide


Upgrade Range
Upgrade Fire Rate
Upgrade Speed
Upgrade Number of Shots
Heal Ship
WASD Movement
Enemy
Enemy Health
Enemies Drop Scrap When they Die
6/100
Number of Kills till Boss spawns
5
Scrap Level
Shallow Water will slow you down
You
P
Press to pause the game
SPACEBAR
Press Space to use your ultimate attack when the bar turns Green
Upgrade Key
Upgrade Level
Q: Heal
4: # Shots - 0/5
3: Fire Rate - 0/5
2: Range - 0/5
1: Speed - 0/5
3 per Upgrade
Collect Scrap To Upgrade Your Ship
V
B
V: Zoom Out B: Zoom In
Shoot Left
Shoot Right
Ultimate Weapon Cooldown
Health