CA 2 ‘Zombie Arena’ Design Document

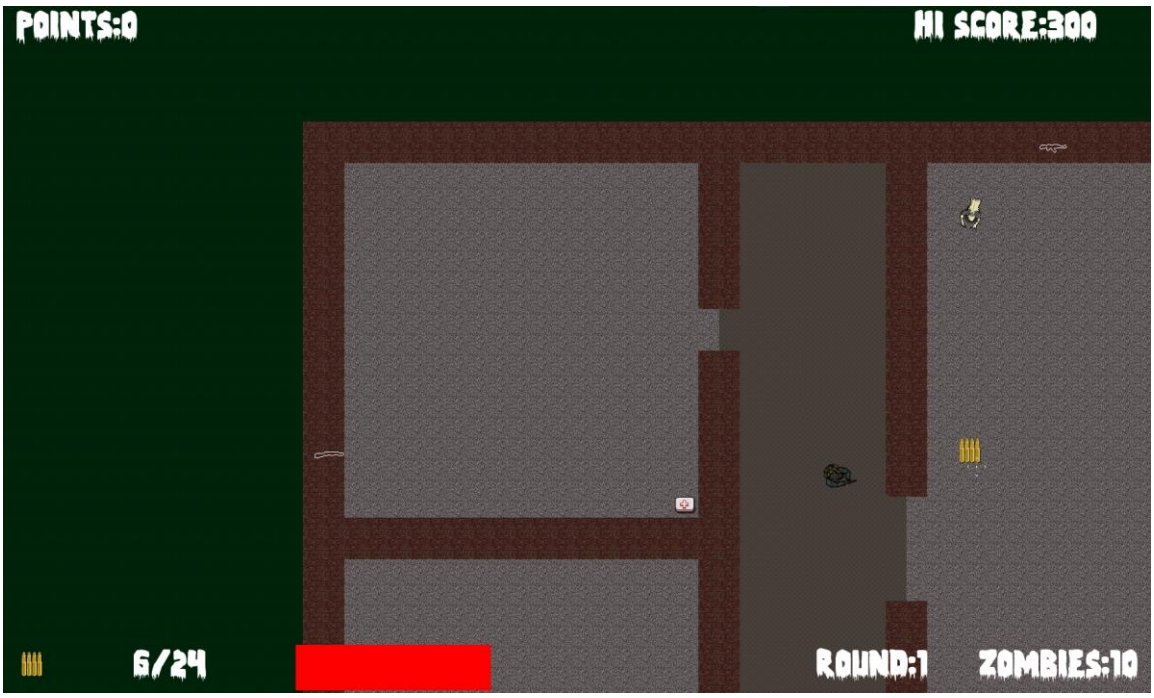
Jake O’ Sullivan

In CA 2 of the Game play programming module, I was tasked with improving the game ‘Zombie Arena’. Zombie Arena is a top-down shooter where the aim of the game is to clear the level of zombies by using a gun with a limited amount of ammunition. Each zombie kill earns the player points, and power ups appear randomly, giving the player more ammunition or more HP. The player moves the character with the arrow keys and aims the gun with the mouse, and the map gets bigger after each level.

What is the game?

In my Improved version of the Zombie Arena game, I changed the game from being level based to round based. Instead of advancing to the next level when all the zombies are defeated, the round number increases. Every time the round increases the difficulty of the game increases as each round has more zombies than the last and the zombie's attributes such as their speed and health are also increased. I also created a map which features separate rooms, making the level more interesting to play through. I added 2 additional weapons; a shotgun and an assault rifle, which can be obtained by spending points gained by defeating zombies.

fig.1 – layout of the map



What do I control?

The player moves the character using the arrow keys and must navigate through the rooms in order to collect power ups and purchase better weapons. The player can aim their weapon by moving the mouse cursor and can fire their weapon by clicking the right mouse button. The player's ammunition is limited so the player must look out for ammunition drops which spawn around the level. The player can purchase new weapons by going up to a weapon icon on the wall and pressing 'E'.

fig.2 - different weapon icons and character models

What is the main focus?

The aim of the game is to survive as many rounds as possible by defeating all of the zombies that spawn each round. To survive as long as possible the player must manage their ammunition usage by carefully lining up their shots, avoiding excessive firing and picking up as many ammo drops as possible. The player must also maneuver around the zombies and plan their movements to avoid being caught in a small room by large groups of zombies.

Describe your approach to coding and implementing the game

The biggest change I made from the original game is the creation of separate rooms in the level. I created the rooms layout by modifying the CreateBackground.cpp file, and creating 5 separate loops, that loop through the vertex array, assigning appropriate wall, or floor tile depending on the shape and size of the room. The first loop fills the vertex array with grass tiles so that the levels exterior will have some textures, then the 4 remaining loops create each of the 4 rooms, replacing some of the wall tiles with floor tiles for the doorways between rooms.

```
for (int w = 50; w < 70; w++) // create room 1
{
    for (int h = 50; h < 70; h++)
    {
        // Position each vertex in the current quad
        rVA[currentVertex + 0].position = Vector2f(w * TILE_SIZE, h * TILE_SIZE);
        rVA[currentVertex + 1].position = Vector2f((w * TILE_SIZE) + TILE_SIZE, h * TILE_SIZE);
        rVA[currentVertex + 2].position = Vector2f((w * TILE_SIZE) + TILE_SIZE, (h * TILE_SIZE) + TILE_SIZE);
        rVA[currentVertex + 3].position = Vector2f((w * TILE_SIZE), (h * TILE_SIZE) + TILE_SIZE);

        if (h >= 59 && h <= 60 && w == 69) // create opening in wall for door
        {
            rVA[currentVertex + 0].texCoords = Vector2f(0, 0 + 0 * TILE_SIZE);
            rVA[currentVertex + 1].texCoords = Vector2f(TILE_SIZE, 0 + 0 * TILE_SIZE);
            rVA[currentVertex + 2].texCoords = Vector2f(TILE_SIZE, TILE_SIZE + 0 * TILE_SIZE);
            rVA[currentVertex + 3].texCoords = Vector2f(0, TILE_SIZE + 0 * TILE_SIZE);
        }
        else if (h == 50 || h == 51 || h == 70 - 1 || w == 50 || w == 51 || w == 70 - 1) // create walls
        {
            rVA[currentVertex + 0].texCoords = Vector2f(0, 0 + 1 * TILE_SIZE);
            rVA[currentVertex + 1].texCoords = Vector2f(TILE_SIZE, 0 + 1 * TILE_SIZE);
            rVA[currentVertex + 2].texCoords = Vector2f(TILE_SIZE, TILE_SIZE + 1 * TILE_SIZE);
            rVA[currentVertex + 3].texCoords = Vector2f(0, TILE_SIZE + 1 * TILE_SIZE);
        }
        else // create floors
        {
            rVA[currentVertex + 0].texCoords = Vector2f(0, 0 + 0 * TILE_SIZE);
            rVA[currentVertex + 1].texCoords = Vector2f(TILE_SIZE, 0 + 0 * TILE_SIZE);
            rVA[currentVertex + 2].texCoords = Vector2f(TILE_SIZE, TILE_SIZE + 0 * TILE_SIZE);
            rVA[currentVertex + 3].texCoords = Vector2f(0, TILE_SIZE + 0 * TILE_SIZE);

        }
```

fig.3 - how room 1 is created in CreateBackground.cpp

To create the boundaries which prevent the player leaving the level and walking through walls, I created a map class which includes 7 different IntRects, 4 for each room and 3 for the doorways, and then set their size to be the exact size of the interior of each room (from wall to wall). I then keep track of the player's position within these IntRects and if the player's position leaves the bounds of any of the IntRects, I revert the player's position to their old position before updating the player object's location.

Here is how the player is kept from walking through the walls of the level:

```
if (!map.getBounds("room", 1).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("room", 2).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("room", 3).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("room", 4).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("door", 1).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("door", 2).contains(m_Position.x, m_Position.y) &&
    !map.getBounds("door", 3).contains(m_Position.x, m_Position.y))
{
    m_Sprite.setPosition(m_PositionLast);
    m_Position = m_PositionLast;
}

m_Sprite.setPosition(m_Position);
```

fig.4 - code to prevent player leaving the map

To prevent the zombies moving straight through the walls and attacking the player I track the current room of the each of the zombies and the player's current room. If a zombie is in a different room than the player, they will head towards the doorway of their current room and enter the middle room. If the player is not in the middle room, the zombie will then check which room the player is in and head towards the doorway that connects with that room. If a zombie and the player are then in the same room, the zombie will simply move towards the player.

I also created a decal class, which allows me to spawn in sprites with textures when something occurs in the game. For example, when a zombie is shot, a blood texture is spawned on the ground giving the player visual feedback that their shot landed. Bullet-holes are also created in the walls whenever a bullet collides with a wall.

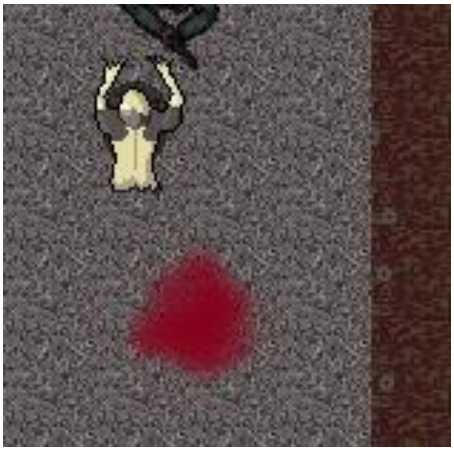


fig.5 - blood splatter and bullet holes

```
void Decal::spawn(String type, float x, float y)
{

    if (type == "bloodImpact")
    {
        m_Sprite = Sprite(TextureHolder::GetTexture(
            "graphics/bloodImpact.png"));
    }
    else if (type == "bulletImpact")
    {
        m_Sprite = Sprite(TextureHolder::GetTexture(
            "graphics/bulletImpact.png"));
    }

    m_Sprite.setPosition(x - 25, y - 25);
```

fig.6 - decal spawning function

Provide appropriate critical analysis and conclusions about developing this game in OO C++:

From creating my version of the Zombie Arena game in C++, I became familiar with how vertex arrays work, and how they can be used with tile-sets to easily create levels with different layouts and textures. I learned how the delete[] operator can be used to ‘destroy’ arrays, deallocating their memory from the heap.