In this Data-Oriented Design assignment, I did it with SDL. I kept some old codes/ classes such as "Window", "InputSystem", and "GameTime" in the engine. As to Rendering, I changed the old class to a *typedef struct*, removed a list storage. So, it only contains basic functions that are needed.

For Entity part, I created a *typedef struct* EntityContainer to save all the properties of units which serves as its name – Container. E.g.

```
typedef struct EntityContainerBase : public RenderBase
{
    short* PositionsX = nullptr;
    short* PositionsY = nullptr;
    short* Widths = nullptr;
    short* Heights = nullptr;
    bool* Usages = nullptr;
    …
    void Init(short maxLength, short maxScreenX, short maxScreenY, short textureWidth,
short textureHeight, const std::string& path);
    void Add(short positionX, short positionY, short widths, short height);
    void BackToPool(short index);
    …
} EntityContainerBase;
```

And it also has functions that are provided for search in corresponded storages(array).

The other approaches for DOD based on "THIS ASSIGENEMTN – Space Shooter" are such as like:

1. Use the smallest data type: Uint8, short, and use pointer* serves as an array, instead of using std::vector<>.
2. Avoid keeping fetch an unlocal data in a for-loop:

   E.g.

```
for (short index = 0; index < enemyContainer.IndexCounter; index++)
{
    short enemyPositionY = enemyContainer.PositionsY[index];
    short laserYOffset = TextureHeight * SpriteIndentOffset;

    if (positionY > enemyPositionY && positionY < (enemyPositionY +
    enemyContainer.TextureHeight * SpriteIndentOffset) || (positionY + laserYOffset) >
    enemyPositionY && (positionY + laserYOffset) < (enemyPositionY +
    enemyContainer.TextureWidth * SpriteIndentOffset))
    {
        …
    }

 }
```

3. Instead of creating a collision class that each entity inherited in the engine, I moved the collision check in the Game instead. Only go with the necessary checks, e.g. the collisions between lasers and the enemies, and the collisions between the player and the enemies.

Due to time limit, I was not able to add sound, texts, and a proper animation. I somehow managed to show an example of animations in the player and an explosion clip. It is not pure Data-Oriented Design, but I tried to reduce as much as possible.