## Intermediary Results

# Models

The models will be box like to keep things simple, e.g. collision detection and to keep the focus on the programming task. To increase the precision of the collision detection, an object could use multiple bounding boxes. This could have a negative effect on performance (more collision detection).

There is the need to read the vertex information from a file into a vertex array.

Every model is represented through a mesh component, which stores all the vertices.  
Meshes and texture are only initialized once and objects uses references of them.

Rotation is stored as matrix.   
This allows easy concatenation of the rotation. The same for quaternions, plus they need less memory space.  
(Essential Mathematics for Games and Interactive Applications, Van Verth James M., third Edition, 2016 Boca Raton S.175)

Every object is in a list/vector, so it can be loop for rendering, updating and collision detection

To render textures on the objects, the shaders need to be adjusted.   
Different algorithms like Bilinear, Trilinear or anisotropic. Antialiasing.

A “PickupManager” manages the spawn of the pickups. It controls where they spawn and what type of pickup spawns.

The skybox is represented by a cube and textured on the inside (Anticlockwise indices). The skybox requires the highest value in the z-buffer, so it is always in the background.

Intersection check:

First check distance between objects, when they are close enough check for intersection

function intersect(a, b) {

return (a.minX <= b.maxX && a.maxX >= b.minX) &&

(a.minY <= b.maxY && a.maxY >= b.minY) &&

(a.minZ <= b.maxZ && a.maxZ >= b.minZ);

)

(<https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection>)

When the surface is flat, we can use 2D Collision Detection (except the projectiles), which is easy to implement.

Movement Check example

public void TurnLeft(GameTime gameTime)

{

float oldRotation = Rotation;

Rotation -= RotationSpeed \* (float)gameTime.ElapsedGameTime.TotalSeconds;

if (CanMove(this.Bounds) == false)

Rotation = oldRotation;

}

public override void MoveForward(GameTime gameTime)

{

// Calculate the new coordinates

float dx = (float)Math.Sin(MathHelper.ToRadians(Rotation)) \* Speed \* (float)gameTime.ElapsedGameTime.TotalSeconds;

float dy = (float)Math.Cos(MathHelper.ToRadians(Rotation)) \* Speed \* float)gameTime.ElapsedGameTime.TotalSeconds;

// create bounding box with new coordinates

Rectangle newBounds = this.Bounds;

newBounds.X = (int)(Position.X + dx);

newBounds.Y = (int)(Position.Y - dy);

// when no collision is detected position is updated to new coordinates

if (CanMove(newBounds))

Position = new Vector2(Position.X + dx, Position.Y - dy);

}

private bool CanMove(Rectangle bounds)

{

for (int i = 0; i < Game.Components.Count; i++)

{

GameObject g = (BasicSpriteComponent)Game.Components[i];

if (g != this) && g.Bounds.Intersects(bounds))

{

return false;

}

}

return true;

}

Explosion/Fire/Dust

Tutorial: <http://www.rastertek.com/dx11tut33.html>  
Shaders which manipulate noise Textures.  
Just set a plane at certain position with a transparent texture, which is always facing towards the player.

Sources:   
Millington Ian, Game Physics Engine Development, Second Edition, Boca Raton 2010  
Van Verth, J.M. and Bishop, L.M. (2008) *Essential mathematics for games and interactive applications: A programmer’s guide*. 2nd edn. Amsterdam: Taylor & Francis.

https://developer.mozilla.org/en-US/docs/Games/Techniques/3D\_collision\_detection