重定ゼミ

『卒業研究論文』

テーマ：３Ｄオンライン対戦型レースゲーム

国際文化学部　国際文化学科
４年Ｇ組　０１Ｇ０６０４
遠藤　正義

目次

序章

・はじめに

私は、卒業研究として「３D オンライン対戦用レース型ゲーム」をテーマに選び、製作致しました。私がこのテーマを選んだ理由としましては、昨今のインターネットの普及により、エンドユーザー同士のネットワーク上での交流や、ゲーム対戦などといったものが非常に活発になってきており、私個人として「オンラインでリアルタイムでの交流」というのが非常に興味深かった為です。その上、今日では個人でも小規模なものであれば製作できることを知り、この製作を機に研究を通じる事で、XML の登場などで更に加速するブロードバンド化に備えた知識を少しでも身につけ、必ず将来において役立つ知識となると考え、オンラインを利用したゲーム製作に至りました。

・制作進行について

私のこの一年間の取り組みの流れとして、私はちょうど一年前に決めてから研究内容は変わっておらず、以下のような流れでこの一年間取り組みました。

DirectX の勉強、簡単な３D のオブジェクトの表示

↓

２D レースゲーム作成（前期終了時点）

↓

ピアチャットの作成にて P2P モデルの理解

↓

３D レースゲーム作成（オフライン）

↓

オンライン３D ゲーム作成

やはり、就職活動などのために最初の方の進行速度はだいぶ遅いですが、後期の特に冬以降の進行速度は自分なりにはかなり速い速度でやれたので、良かったという風に思います。

・　製作概要

ネットワークモデル P2P モデル（後に説明）を利用し、最初にホストを起動させたプレイヤーとその他二人のプレイヤーで一つのセッションとなり、レースシーンではキー連打のみの簡易な操作で、3D で作られた使用キャラクター（自動車）が前方向のみに進み、キーの押した回数によって速度が変化し、リアルタイムで対戦できるといったものです。

・　環境について

使用した環境は以下の通りです。

＜VisualBasic.net＞（以下 VB）

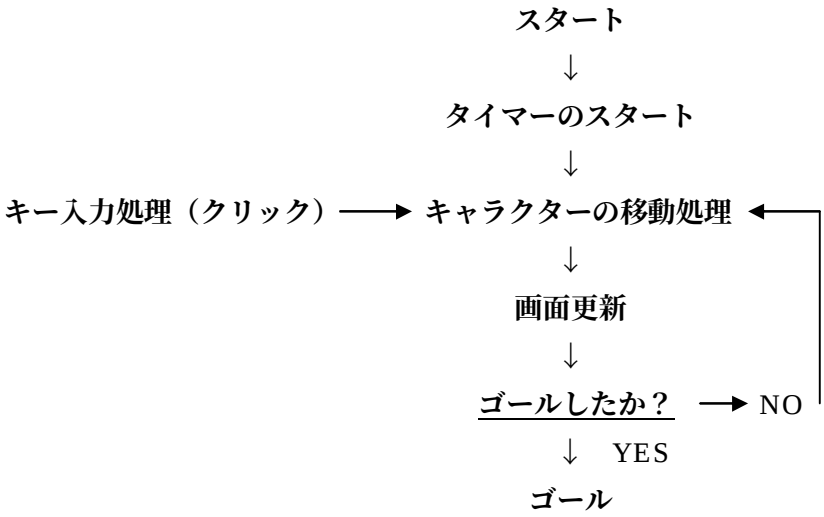
＜MetasequoiaLE＞（以下 Metasequoia）
＜DirectX9.0＞
VB は勿論の事、システム全般であり、Metasequoia はネット上からダウンロードする事ができる３DCG 作成用ソフトのフリー版です。研究に VB を選んだ理由としては、やはりゼミ三年時の活動の頃より非常に馴染みの深い言語であり、また、私のようなゲーム作成の場合は特に向いている言語である為です。一方、Metasequoia は参考文献内で使われていたソフトであり、比較的簡単に３DCG を作成できるという事で挑戦致しました。そして、DirectX9.0 においては.Net 環境下での３DCG の描画と、API の DirectPlay によってネットワークゲーム作成の簡易化を可能にしています。

一章　２D レースゲームについて

・　２D レースゲームの流れ

私は最初に、簡単な３D オブジェクトの表示に取り掛かりました。しかし、この部分は３D レースゲームシーンと関わりがあるので、後の段落にて述べるものとします。そして、この段落では私が夏に取り組んでいた２D レースゲームについて述べます。

まず、ゲームの概要ですが、操作方法などのインターフェース面においての２D であること以外は何も３D のものと変わりません。プログラミングの流れは以下のようになります。



・ソース解説

このゲームに関しては、非常に簡単なものなので、改良したTimerクラスと敵プレイヤーの移動処理を主に解説いたします。まずはCPUとの対戦ではなく、一人用ゲームのソースを用います。

＜SOURCE＞（抜粋）

```
Private WithEvents timer_ As GameTimer 'ゲームの経過時間
  Private clickCount As Integer = 0 'クリックした回数

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    ' ボタンをクリックした回数をカウントする
    clickCount = clickCount + 1
  End Sub
  Private Sub gameTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles timer_.Tick
    ' 速度・経過時間の計算、表示
    speedMaterLabel.Text = "速度： " & CInt(clickCount / (timer_.GetTime() / 1000))
& " クリック/秒"
    timeLabel.Text = "経過時間： " & CInt(timer_.GetTime / 1000) & " 秒"
    ' 座標の移動
    myPlayer.Location = New System.Drawing.Point(clickCount * 5,
myPlayer.Location.Y)
```

解説

キャラクターは１クリックにつき５ピクセル動き、ゴールは当たり判定によってゴールとなります。

```
' ゴールの判定
    If myPlayer.Location.X > 424  - 64 Then
      timer_.Stop() 'ゲームタイマーの停止
      MessageBox.Show("ゴール！")
    End If
' ゲーム内の時間を制御するタイマー
Public Class GameTimer
  Inherits System.Windows.Forms.Timer
  Private startTime_ As Long = 0
```

解説

Inheritsステートメントを使い、Timerクラスを継承した新しいクラスGametimer を作り、ゲーム開始から何秒経ったのかを返すGetTimeメソッド追加しています。また、経過した時間を計算するにはOSが起動してから何秒経過したのかを取得するプロパティの

System.Environment.Tickcountを利用しています。

```
  Public Sub New()
    Reset()
    Me.Stop()
  End Sub

  ' 現在の時間を0に初期化する
  Public Sub Reset()
    startTime_ = System.Environment.TickCount
  End Sub
  ' Resetしたときから何ミリ秒経ったかを取得する
  Public Function GetTime() As Integer
    Return System.Environment.TickCount  - startTime_
  End Function
End Class
```

・ CPU 対戦版

下記が対戦版のソースの抜粋です。タイマー処理は全く同じですが、CPU がいるため、新しくプレイヤークラスを作り、自分のキャラクターを Plyer(0)とし、Clickcount 変数を使って計算しています。

```
Private WithEvents timer_ As GameTimer 'ゲームの経過時間
  'Private clickCount As Integer = 0 'クリックした回数
  Private player_(3) As Player ' 自分、敵のステータス情報
  Private random_ As random = New random()
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    ' ボタンをクリックした回数をカウントする
    player_(0).ClickCount = player_(0).ClickCount + 1
  End Sub
  Private Sub gameTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles timer_.Tick
```

解説

敵プレイヤーの移動処理には、画面の更新が行われるたびに一歩進んだり進まないように

なっています。また、座標の移動とゴール判定には自分と敵プレイヤー二つの三つに対してForループを利用して処理しています。

```
    ' 敵キャラクターの移動処理
    Dim i As Integer
    For i = 1 To 2
      player_(i).ClickCount = player_(i).ClickCount + random_.NextDouble()
    Next
    ' 速度・経過時間の計算、表示
    speedMaterLabel.Text = "速度： " & CInt(player_(0).ClickCount * 1000 / timer_.GetTime()) & " クリック/秒"
    timeLabel.Text = "経過時間： " & CInt(timer_.GetTime / 1000) & " 秒"
    ' 座標の移動
    For i = 0 To 2
      player_(i).Picture.Location = New System.Drawing.Point(player_(i).ClickCount * 5, player_(i).Picture.Location.Y)
    Next

    ' ゴールの判定
    For i = 0 To 2
      If player_(i).Picture.Location.X > 424  - 64 Then
        timer_.Stop() 'ゲームタイマーの停止
        MessageBox.Show("ゴール！")
      End If
    Next
  End Sub
```

以上が２Ｄレースゲームの解説になります。

二章　オフライン３Ｄレースゲーム

・ このプログラムの３ＤＣＧについて

この章では３ＤＣＧについての解説を中心に、作成したオフライン３Ｄレースゲームについて解説致します。

まず確認として、冒頭でも述べましたが、私は Metasequoia にて、マウス操作のみでできる比較的簡易な操作にて、このプログラムに使われているＣＧを作成致しました。し

かし、この私のプログラムの場合、作成した３Ｄオブジェクト（以下モデルデータ）自身はアニメーション致しません。私が必要としたのは、座標移動処理のみを行うものなので、アニメーションを使わないモデルデータ自体のみでした。そして、プログラムには拡張子を＜.x＞とし、Ｘファイル（モデルデータ）として使用しています。ちなみに補足として、私が使用した Metasequoia というソフトは、まさにモデルデータを作成できるが、アニメーションをさせる事ができないというもので、アニメーションをさせる場合は他のソフトへそのままのモデルデータを変換して受け渡すことでできるようになります。

　また、３ＤＣＧの作成等についての詳細は、今回のプログラミングの趣旨と大分変わってしまうので、ここで、プログラム解説にあたって必要だと思われる３ＤＣＧについての知識を簡潔に説明します。

モデリング…コンピュータの中で物体を作成する作業の事

レンダリング…配置、質感設定などの完成画像の仕上げ作成

ライティング…３Ｄにおいて光源はなくてはならないもの
　　ex.平行光源（Directional）点光源（Point）

テクスチャ…色などの貼り付けた模様のこと

カメラの設定…「カメラの位置」、「カメラの向いている方向」、「DirectX の空間で上を示す方向」の３つを指定する。Matrix 構造体には LookAtLH という関数が用意されている。

座標系　…　３ＤではＸ座標（幅）、Ｙ座標（高さ）のほかにＺ座標（奥行き）が必要。また、三次元の中で物体の位置を特定するために、DirectX では通常「左手座標系（左手の親指、人差し指、中指を直角に伸ばし、それぞれをＸＹＺ軸に見立てる）」が使われている。

行列の変換　…　３ＤＣＧでは、その特性を活かした移動、回転、拡大・縮小といった変換作業を良く使い、このような計算に「行列（マトリックス）」を使う。
DirectX
で計算する場合は Matrix 構造体にメソッドが用意されているので、複雑な計算は通常は不要。

投影変換…物体の距離などから計算された投影。つまり遠近法。Matrix 構造体には投影変

換のための関数 PerspectiveFovLH がある。

ポリゴンとメッシュ…３ＤＣＧを表現しているものがポリゴンであり、そのポリゴンを集めたものを「メッシュ（網）」といい、一つのグループで表現している。

頂点…３D オブジェクトを構成しているポリゴンの接点

境界球…３D オブジェクトのメッシュが在る領域であり、オブジェクト全頂点を含む最小の球体

マテリアル…物体の色や質感などの基本的な情報のこと。以下の５項目が設定できる。

- Ambient　　環境光の色
- Diffuse　　拡散反射光の色
- Emissive　　自身の発光する色
- SpecularSharpness　　鏡面反射光の大きさ
- Specular　　鏡面反射光の色

以上のことを簡単ではありますが、踏まえたうえでオフライン３Ｄレースゲームについて次の段落で解説していきます。

- ３Ｄレースゲーム（オフライン）について

まず、以下が使用しているクラス全てになります。

<DirectXPanel>　　DirectX の表示を管理するクラス
<GameTimer>　　ゲーム内の時間を管理するクラス
<MeshObject>　　メッシュの形状データ、マテリアル、テクスチャを管理するクラス
<MeshObjectList>　　複数の MeshObject オブジェクトを管理するクラス
<Player>　　シーンの基盤となるインターフェース
<PlaerList>　　レースゲーム本体のシーンクラス
<SceneBase>　　シーンの基底となるインターフェース
<GameMainScene>　　レースゲーム本体のシーンクラス
<GameTitleScene>　　タイトル画面のシーンクラス
<GameOverScene>　　ゲームオーバー画面のシーンクラス
<RaceForm>　　レースゲーム全体を管理するクラス

このゲームではタイトル画面、レースゲームシーン、ゲームオーバー画面の３つのシー

ンに分かれており、全てのシーンに共通する部分を<ScenBase>インターフェイスとして定義しています。

```
Public Interface SceneBase

  Sub MoveTo()
  Sub PaintTo()
End Interface
```

MoveTo メソッドは、キャラクターの移動に関する処理を行うメソッドとして利用し、PaintTo メソッドは画面の描画に関する処理を行うメソッドとして利用します。両方ともタイマー制御によって一定時間ごとに呼び出されるようにしているわけですが、二つに分けた理由は３DCG の描画は PC によっては荷が重い処理のため、時間内に終わる保障がないためです。そのため、MoveTo メソッドでは毎回処理しなければならない座標移動などの処理を記述し、負荷の高い描画処理を PaintTo メソッドに記述してマシンのスペックに合わせて呼び出すタイミングを制御している為です。

次にタイトルシーンの説明です。ここでは DirectX を使っておらず、リアルタイムに処理をする必要がないので、MoveTo、PaintTo メソッドも何も処理を行いません。StartButton をクリックすると、レースゲームシーンに切り替えるメソッド GameStart を呼び出します。

```
Private main As RaceForm
  Public Sub New(ByRef main As RaceForm)
    MyBase.New()
    ' この呼び出しは Windows フォーム デザイナで必要です。
    InitializeComponent()
    Me.main = main
  End Sub

  Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartButton.Click
    main.GameStart()
  End Sub

  Private Sub SceneBase_MoveTo() Implements SceneBase.MoveTo
  End Sub

  Private Sub SceneBase_PaintTo() Implements SceneBase.PaintTo
```

```
  End Sub
End Class
```

次に、シーン移動の制御を行う<RaceForm>クラスのソースコードです。このクラス一つでscene_インスタンスの中身を表示したいシーンに応じて変更し、シーンやタイマーの制御をしています。

```
' ゲームのシーン管理を行う。
Public Class RaceForm
  Inherits System.Windows.Forms.Form
Friend WithEvents gameTitleScene As GameTitleScene
  Friend WithEvents gameMainScene As gameMainScene
  Friend WithEvents gameOverScene As GameOverScene
  Private WithEvents gameTimer_ As GameTimer

  Private scene_ As SceneBase
  Private directXPanel_ As DirectXPanel
```

上記のDirectXPanelについては後述いたします。

```
  Public Sub GameTitle()
    If Me.Controls.Contains(gameOverScene) = True Then
      Me.Controls.Remove(gameOverScene)
      gameOverScene.Dispose()
    End If
    gameTitleScene = New GameTitleScene(Me)
    Me.Controls.Add(gameTitleScene)
    scene_ = gameTitleScene
  End Sub
```

以下が先述したGameStartメソッドになります。

```
  Public Sub GameStart()
    If Me.Controls.Contains(gameTitleScene) = True Then
      Me.Controls.Remove(gameTitleScene)
      gameTitleScene.Dispose()
```

```
    End If
    If Not directXPanel_ Is Nothing Then directXPanel_.Dispose()
    directXPanel_ = New DirectXPanel()

    Me.gameMainScene = New RaceGame.GameMainScene(Me, directXPanel_)
    Me.Controls.Add(gameMainScene)

    scene_ = gameMainScene

    gameTimer_ = New GameTimer()
    gameTimer_.Start()
  End Sub

  Public Sub GameOver()
    If Me.Controls.Contains(gameMainScene) = True Then
      ' gameMainSceneをDisposeする前に、DirectXPanelをRemoveして、いっしょに
Disposesされないようにする。
      gameMainScene.Controls.Remove(directXPanel_)
      Me.Controls.Remove(gameMainScene)
      gameMainScene.Dispose()
    End If

    gameOverScene = New GameOverScene(Me, directXPanel_)
    Me.Controls.Add(gameOverScene)
    gameOverScene.Visible = True
    scene_ = gameOverScene
  End Sub

  Private Sub GameTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles gameTimer_.Tick
    'Debug.WriteLine("RaceForm.GameTimerTick")
    scene_.MoveTo()
    scene_.PaintTo()
  End Sub

  Private Sub RaceForm_Load(ByVal sender As Object, ByVal e As System.EventArgs)
```

```
Handles MyBase.Load
    directXPanel_ = New DirectXPanel()
    Me.GameTitle()
  End Sub
End Class
```

・ DirectXPanelクラスについて

ここではDirectXPanelクラスについての解説を致します。例えば、シーンの移り変わりを考えた場合、レースシーンからゴールシーンに変わるときにメッシュの描画作成をまったく最初からするのでは計算時間がかかってしまいます。そのため、前のシーンからメッシュを引き継いで次のシーンに移れる手段が必要となり、そこでメッシュ、DirectXの動作に必要なオブジェクト等を管理するクラスを用意しました。それがDirectXPanelクラスになります。

全ソースコード

```
Public Class DirectXPanel
  Inherits System.Windows.Forms.UserControl
' DirectXを使うのに必要なインスタンス
  Protected direct3DDevice_ As Device

  ' DirectX Graphics の初期化
  Protected Overridable Sub InitializeDirect3D()
    ' Direct3DDeviceを取得
    direct3DDevice_ = CreateDirect3DDevice(CreateDefaultPresentParameters())
  End Sub

  Private Function CreateDirect3DDevice(ByVal presentParameters As
PresentParameters) As Device
    ' Direct3DDeviceクラスのインスタンスを生成
    Dim result As Device
    Try
      result = New Device(0, DeviceType.Hardware, Me,
CreateFlags.HardwareVertexProcessing, presentParameters)
      Debug.WriteLine("Created HAL & HVP mode.")
    Catch
      Try
        result = New Device(0, DeviceType.Hardware, Me,
```

```
CreateFlags.SoftwareVertexProcessing, presentParameters)
        Debug.WriteLine("Created HAL mode.")
      Catch
        Try
          result = New Device(0, DeviceType.Reference, Me,
CreateFlags.SoftwareVertexProcessing, presentParameters)
          Debug.WriteLine("Created REF mode.")
        Catch
          MessageBox.Show("このPCはDirect3Dを利用できません。")
          Debug.WriteLine("このPCはDirect3Dを利用できません。")
          result = Nothing
        End Try
      End Try
    End Try
    Return result
  End Function

  ' 描画時の設定値PresentParameters構造体を初期化する
  Private Function CreateDefaultPresentParameters() As PresentParameters
    Dim result As New PresentParameters()
    With result
      .Windowed = True
      .SwapEffect = SwapEffect.Discard
      .EnableAutoDepthStencil = True
      .AutoDepthStencilFormat = DepthFormat.D16
    End With
    Return result
  End Function


  ' 描画メッシュの初期化
  ' メッシュに、Xファイルから読み込んだ情報を設定する
  Protected meshObjectList_ As MeshObjectList = New MeshObjectList()
  Public Function GetMeshObjectList() As MeshObjectList
    Return meshObjectList_
  End Function
```

```
  Protected Overridable Sub AddMeshObject(ByVal meshObject As MeshObject)
    meshObjectList_.AddMeshObject(meshObject)
  End Sub

  ' 再描画のたびにバックグラウンドカラー(標準は灰色)で塗りつぶす処理をしないようにする。
  Protected Overrides Sub OnPaintBackground(ByVal e As System.Windows.Forms.PaintEventArgs)
  End Sub

  ' 描画の処理
  Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    ' 画面を初期化
    direct3DDevice_.Clear(ClearFlags.Target Or ClearFlags.ZBuffer, Color.Blue, 1.0F, 0)

    '描画指示の開始を宣言する
    direct3DDevice_.BeginScene()
    ' 描画する物体を生成して、描画を指示する

    'Debug.WriteLine("DirectXPanel.OnPaint:meshobject count=" & meshObjectList_.Count)
    Dim i As Integer
    For i = 0 To meshObjectList_.Count  - 1
      ' 描画中にプレイヤーが退場して例外が発生することがあります。(オンラインの場合)
      Try
        'Debug.WriteLine("  DirectXPanel.OnPaint:meshobject filename=" & meshObjectList_.GetMeshObject(i).Filename)
        If meshObjectList_.GetMeshObject(i) Is Nothing Then Exit For
        direct3DDevice_.Transform.World = meshObjectList_.GetMeshObject(i).MeshMatrix
        Dim j As Integer
        For j = 0 To meshObjectList_.GetMeshObject(i).MeshMaterials.Length  - 1
          direct3DDevice_.Material =
```

```
meshObjectList_.GetMeshObject(i).MeshMaterials(j)
          direct3DDevice_.SetTexture(0,
meshObjectList_.GetMeshObject(i).MeshTextures(j))
          meshObjectList_.GetMeshObject(i).MeshValue.DrawSubset(j)
        Next j
      Catch ex As Exception
        Debug.WriteLine(ex.Message & vbCrLf & ex.StackTrace)
      End Try
    Next i

    '描画指示の終了を宣言する
    direct3DDevice_.EndScene()
    '画面を最新の状態に更新する
    direct3DDevice_.Present()
  End Sub

  ' シーンの初期化
  Protected Overridable Sub InitializeScene()
    Debug.WriteLine("Initialize Scene.")
    ' レンダリング時のオプションを指定
    direct3DDevice_.RenderState.ZBufferEnable = True

    ' カメラの設定
    direct3DDevice_.Transform.View = Matrix.LookAtLH(New Vector3(5, 5, -10), New
Vector3(0, 0, 0), New Vector3(0.0F, 1.0F, 0.0F))
    ' 射影の設定
    direct3DDevice_.Transform.Projection = Matrix.PerspectiveFovLH(CSng(Math.PI /
2), 1.3F, 1.0F, 100.0F)
    ' 照明の設定

    direct3DDevice_.RenderState.Lighting = True
    direct3DDevice_.Lights(0).Type = LightType.Directional
    direct3DDevice_.Lights(0).Direction = New Vector3( -1, -1, -1)
    direct3DDevice_.Lights(0).Diffuse = Color.FromArgb(255, 255, 255)
    direct3DDevice_.Lights(0).Commit()
    direct3DDevice_.Lights(0).Enabled = True
```

```
  End Sub

  Public Function GetDirect3DDevice() As Device
    Return direct3DDevice_
  End Function
```

下記では、新たにメッシュを描画しています。

```
  ' ウィンドウのサイズ変更やシーンの移動などでDeviceが破棄されたとき、
  ' 新しくDeviceを作り直す。同時にメッシュも作り直す。
  Public Sub RefleshDevice()
    InitializeDirect3D()
    Dim i As Integer
    For i = 0 To Me.meshObjectList_.Count  - 1
      meshObjectList_(i).RefleshDevice(Me.direct3DDevice_)
    Next
    InitializeScene()

  End Sub

End Class
```

また、上記のように DirectXPanel クラスはその内部にメッシュオブジェクトの情報を持っており、必要に応じてメッシュを描画します。しかし、メッシュを描画するには Material や Texture などの形状データなどのいろいろなデータを扱うため、一つの物体を描画するのに必要な情報をまとめて以下の MeshObject クラスにまとめます。ここでは SceneBace クラスと同じように、MoveTo クラスを持っており、その中で座標変換の処理を記述して、一箇所で全ての座標変換をしなくてもすむように制御しています。

```
Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D

' メッシュの管理を行う。
Public Class MeshObject
  ' 形状データ
  Protected mesh_ As Mesh
  Property MeshValue() As Mesh
```

```
  Get
    Return mesh_
  End Get
  Set(ByVal Value As Mesh)
    mesh_ = Value
  End Set
End Property

' 質感データ
Protected meshMaterials_() As Material
Property MeshMaterials() As Material()
  Get
    Return meshMaterials_
  End Get
  Set(ByVal Value As Material())
    meshMaterials_ = Value
  End Set
End Property

' テクスチャデータ
Protected meshTextures_() As Texture
Property MeshTextures() As Texture()
  Get
    Return meshTextures_
  End Get
  Set(ByVal Value As Texture())
    meshTextures_ = Value
  End Set
End Property

' 座標データ
Protected meshMatrix_ As Matrix = Matrix.Translation(0, 0, 0)
Property MeshMatrix() As Matrix
  Get
    Return meshMatrix_
  End Get
```

```
    Set(ByVal Value As Matrix)
      meshMatrix_ = Value
    End Set
  End Property

  ' 境界球の半径
  Private boundingSphere_ As Single
  Public ReadOnly Property BoundingSphere() As Single
    Get
      Return boundingSphere_
    End Get
  End Property

  Public Filename As String
  ' XファイルからMeshObjectを生成する
  Public Sub New(ByVal filename As String, ByRef direct3DDevice As Device)
    Me.Filename = filename
    If Not direct3DDevice Is Nothing Then
      Me.RefleshDevice(direct3DDevice)
    End If
  End Sub

  Public Sub RefleshDevice(ByRef direct3DDevice As Device)
    Debug.WriteLine("MeshObject.RefleshDevice")
    Dim extMaterials As ExtendedMaterial()
    Dim mesh As Mesh = mesh.FromFile(Filename, MeshFlags.Managed, direct3DDevice,
extMaterials)
    Dim meshMaterials As Material()
    ReDim meshMaterials(extMaterials.Length)
    Dim meshTextures As Texture()
    ReDim meshTextures(extMaterials.Length)

    Dim i As Integer
    'Materialマテリアルの読み込み
    For i = 0 To extMaterials.Length  - 1
      meshMaterials(i) = extMaterials(i).Material3D
```

```
      ' Texture（テクスチャ）ファイルの読み込み
      meshTextures(i) = Nothing
      Dim textureFilename As String = extMaterials(i).TextureFilename
      If Not textureFilename Is Nothing Then
        meshTextures(i) = TextureLoader.FromFile(direct3DDevice, textureFilename)
      End If
    Next

    mesh_ = mesh
    meshMaterials_ = meshMaterials
    meshTextures_ = meshTextures
    boundingSphere_ = ComputeBoundingSphere(New Vector3(0, 0, 0))
  End Sub

  ' 座標変換
  Public Overridable Sub MoveTo()
  End Sub

  ' 境界球の計算
  Public Function ComputeBoundingSphere(ByVal objectCenter As Vector3) As Single
    Dim i32BitFlag As MeshFlags
    Dim result As Single = 0.0F

    If mesh_.Options.Use32Bit = True Then
      i32BitFlag = MeshFlags.Use32Bit
    End If

    ' 頂点バッファを取得
    Dim vertexBuffer As VertexBuffer = mesh_.VertexBuffer

    ' 境界球を作成
    Dim vertexData As GraphicsStream = vertexBuffer.Lock(0, 0,
LockFlags.NoSystemLock)
    result = Geometry.ComputeBoundingSphere(vertexData, mesh_.NumberVertices,
mesh_.VertexFormat, objectCenter)
    vertexBuffer.Unlock()
```

```
    vertexBuffer.Dispose()
    Return result
  End Function


  Public Sub Dispose()
    Me.mesh_.Dispose()
  End Sub
End Class
```

また、これに関連して、上記の MeshObject クラスを複数管理するクラスとして、二次元配列を表す ArrayList クラスを継承した MeshObjectList クラスを下記のように作ります。

```
' 複数のMeshObjectを管理する
Public Class MeshObjectList
  Inherits System.Collections.ArrayList

  Public Sub AddMeshObject(ByRef meshObject As MeshObject)
    Me.Add(meshObject)
  End Sub

  Public Function GetMeshObject(ByVal index As Integer) As MeshObject
    Return Me(index)
  End Function

  Public Sub RemoveMeshObject(ByRef meshObject As MeshObject)
    Debug.WriteLine("MeshObjectList.RemoveMeshObject")
    Me.Remove(meshObject)
  End Sub
End Class
```

ここまでで述べたように、以上の DirectXpanel クラス、MeshObject クラス、MeshObjectList クラスはレースゲームとは全く関係のない、DirectX 独自の処理を行うクラスとして用意しました。

・ キャラクター管理について

キャラクタープレイヤーは赤、、青、黄の簡単な自動車を Metasequoia にて作りました。それぞれのキャラクターは形状データ MeshObject オブジェクトと、いくつかのプロパティ

を持っており、以下が Player クラスになります。

Player クラスのプロパティ

名称	型	説明
name	string	キャラクターの名前（色）
position	Vector3	キャラクターの座標
speed	Single	動く速度
ViewPosition	Vector3	自分を視点にする時の視点の位置
ViewTarget	Vector3	自分を視点にするときの視点の注意対象

```
Public Class Player
  Inherits MeshObject

  ' キャラクターの種類
  Public Property CharactorType() As String
    Get
      Dim path() As String = MyBase.Filename.Split("\")
      Return path(path.Length  - 1)
    End Get
    Set(ByVal Value As String)
      Dim path() As String = MyBase.Filename.Split("\")
      Dim filename As String
      Dim i As Integer
      For i = 0 To path.Length  - 2
        filename += path(i) & "\"
      Next
      filename += Value
      Debug.WriteLine("Player.CharactorType:value=" & Value & "filename=" &
filename)
      MyBase.Filename = filename
    End Set
  End Property

  Private name_ As String
  Public Property Name() As String
```

```
  Get
    Return name_
  End Get
  Set(ByVal Value As String)
    name_ = Value
  End Set
End Property

Public MAX_SPEED As Single = 1.0F
Private speed_ As Single
Public ReadOnly Property Speed() As Single
  Get
    Return speed_
  End Get
End Property
```

オブジェクトの座標

```
Protected position_ As Vector3
Public Property Position() As Vector3
  Get
    Return position_
  End Get
  Set(ByVal Value As Vector3)
    position_ = Value
  End Set
End Property

Public Shadows Property MeshMatrix() As Matrix
  Get
    Me.MoveTo()
    Return meshMatrix_
  End Get
  Set(ByVal Value As Matrix)
    meshMatrix_ = Value
  End Set
```

```
  End Property

  自分を視点にするときの、視点の位置

  Public ReadOnly Property ViewPosition() As Vector3
    Get
      '自分のキャラクターのすこし後ろにして、画面の中心に自分のキャラクターの後姿
が見えるようにする
      Dim result As Vector3 = New Vector3()
      result.X = position_.X  - 3 * Math.Cos(angle_)
      result.Y = position_.Y + 1
      result.Z = position_.Z  - 3 * Math.Sin(angle_)
      Return result
    End Get
  End Property

  Private angle_ As Single
  ' 自分を視点にするときの、視点の注視対象
  Public ReadOnly Property ViewTarget() As Vector3
    Get
      '  自分自身を注視するように
      Dim result As Vector3 = New Vector3(position_.X, position_.Y + 1, position_.Z)
      Return result
    End Get
  End Property

  Public Sub New(ByVal name As String, ByVal filename As String, ByVal direct3DDevice
As Device, ByVal position As Vector3)
    MyBase.New(filename, direct3DDevice)
    name_ = name
    position_ = position
    Initialize()
  End Sub

  Public Sub Initialize()
    angle_ = Math.PI * 1 / 2
```

```
    speed_ = 0.0F
  End Sub

  Public Sub Turn(ByVal angle As Single)
    '
    angle_ = angle_ + angle
    If angle_ > Math.PI Then angle_ = angle_  - Math.PI * 2
    If angle_ <  -Math.PI Then angle_ = angle_ + Math.PI * 2
  End Sub

  Public Sub SpeedUp(ByVal speed As Single)
    'Debug.WriteLine("Player.SpeedUp")
    speed_ = speed_ + speed
    If speed_ > MAX_SPEED Then speed_ = MAX_SPEED
    If speed_ < 0 Then speed_ = 0
  End Sub

  Public Overrides Sub MoveTo()

    Dim moveMatrix As Matrix

    'はじめにRotation
    Rotasionの回転角の指定は、時計回りを正とするので、angleと符号が逆になります。
    moveMatrix = Matrix.RotationY( -(angle_ + Math.PI * 1 / 2))
```

以下ではスピードが加わったときのキャラクターの座標を修正しています。

```
    '現在のスピードと方向に従って位置を修正
    'x -z平面に対して計算することに注意。
    position_.X = position_.X + speed_ * Math.Cos(angle_)
    position_.Z = position_.Z + speed_ * Math.Sin(angle_)
    'つぎにTranslation
    moveMatrix.Multiply(Matrix.Translation(position_))

    meshMatrix_ = moveMatrix
    ' Debug.WriteLine("Player.MoveTo:" & Me.Name & " position:" &
```

```
position_.ToString() & " speed=" & speed_ & " angle=" + angle_.ToString())
  End Sub
End Class
```

また、MeshObject クラスと同じように複数管理するために PlayerList クラスを用意します。AllayList クラスを継承している事など、ほぼ同じですが、MeshObjectList クラスと関係付けて作られています。

```
Public Class PlayerList
  Inherits System.Collections.ArrayList

  Private meshObjectList_ As MeshObjectList
  Public Sub New(ByRef meshObjectList As MeshObjectList)
    meshObjectList_ = meshObjectList
  End Sub

  Public Sub AddPlayer(ByRef player As Player)
    Me.Add(player)
    meshObjectList_.AddMeshObject(player)
  End Sub

  Public Function GetPlayer(ByVal index As Integer) As Player
    Return Me(index)
  End Function

  Public Sub RemovePlayer(ByRef player As Player)
    Me.Remove(player)
    meshObjectList_.RemoveMeshObject(player)
  End Sub

End Class
```

・ ゲームシーンについて

以下はゲームのメインとなるシーンのソースコードです。全体の処理の流れは以下のようになります。

１、RaceForm オブジェクト、DirectPanel オブジェクトの関連付け。

２、コンポーネントの初期化
３、キャラクターの初期化、配置
４、コースの初期化、配置

```
Public Class GameMainScene
  Inherits System.Windows.Forms.UserControl
  Implements SceneBase
Private main As RaceForm
  Private directXPanel As DirectXPanel

  Public Sub New(ByRef main As RaceForm, ByRef directXPanel As directXPanel)
    Me.main = main
    Me.directXPanel = directXPanel

    ' コンポーネントの初期化
    InitializeComponent()

    'Controlの追加
    Me.Controls.Add(Me.directXPanel)

    InitializePlayerList()
    InitializeCourse()
  End Sub

  Private playerList_ As PlayerList
  Private myPlayer_ As Player
  Private enemy1_ As Player
  Private enemy2_ As Player
  Private Sub InitializePlayerList()
    Debug.WriteLine("GameMain.InitializePlayerList")
    playerList_ = New PlayerList(directXPanel.GetMeshObjectList())
```

以下はプレイヤー、配置座標、最大スピード。Turnというのは初期状態ではキャラクターが逆を向いているので１８０度向きを変えています。

```
    myPlayer_ = New Player("user", Application.ExecutablePath &
```

```
"\..\..\..\medias\red.x", directXPanel.GetDirect3DDevice(), New Vector3(0, 1, 0))
    myPlayer_.Turn(Math.PI)
    myPlayer_.MAX_SPEED = 1.0F
    enemy1_ = New Player("enemy1", Application.ExecutablePath &
"\..\..\..\medias\blue.x", directXPanel.GetDirect3DDevice(), New Vector3( -3, 1, 0))
    enemy1_.Turn(Math.PI)
    enemy1_.MAX_SPEED = 1.5F
    enemy2_ = New Player("enemy2", Application.ExecutablePath &
"\..\..\..\medias\yellow.x", directXPanel.GetDirect3DDevice(), New Vector3(3, 1,
0))
    enemy2_.Turn(Math.PI)
    enemy2_.MAX_SPEED = 2.0F

    playerList_.AddPlayer(myPlayer_)
    playerList_.AddPlayer(enemy1_)
    playerList_.AddPlayer(enemy2_)

  End Sub
```

以下はコースとゴールフラグの配置になります。

```
  Private course_ As MeshObject
  Private Sub InitializeCourse()
    course_ = New MeshObject(Application.ExecutablePath &
"\..\..\..\medias\course.x", directXPanel.GetDirect3DDevice())
    directXPanel.GetMeshObjectList().AddMeshObject(course_)
  End Sub
```

MoveToメソッドはRceFormクラスから一定時間毎に呼び出されます。そのたびに時間を進行させ、ゴールの判定を行います。

```
  ' ゲームループ
  Private Const TIME_LIMIT As Integer = 1000
  Private time_ As Integer = 0
  Private random_ As Random = New Random()
  Public Sub SceneBase_MoveTo() Implements SceneBase.MoveTo
```

```
    time_ = time_ + 1
    myPlayer_.SpeedUp( -0.05)
    '制限時間を越えていたらゲームオーバー
    If time_ > TIME_LIMIT Then
      TimeUp()
    End If
    timeLabel.Text = String.Format("TIME {0}", time_)
    speedLabel.Text = String.Format("SPEED {0} km/h", CInt(myPlayer_.Speed * 100))
```

以下は敵プレイヤーの変化する速度ですが、内容は２Ｄレースゲームと全く変わりません。

```
    '敵のスピードをランダムに変化させる
    enemy1_.SpeedUp((random_.NextDouble  - 0.2) / 30)
    enemy2_.SpeedUp((random_.NextDouble  - 0.3) / 20)

    'Playerの座標変換を行う
    Dim i As Integer
    For i = 0 To playerList_.Count  - 1
      If playerList_.GetPlayer(i) Is Nothing Then Exit For
      playerList_.GetPlayer(i).MoveTo()
    Next i

    ' ゴール地点に到達したかどうかをチェック
    For i = 0 To playerList_.Count  - 1
      If playerList_.GetPlayer(i) Is Nothing Then Exit For
      If playerList_.GetPlayer(i).Position.Z <  -52 Then
        main.GameOver()
      End If
    Next i
  End Sub
```

下記のPaintToメソッドでは自分のキャラクターの後方に視点（カメラ）を合わせるようにしています。

```
  Public Sub SceneBase_PaintTo() Implements SceneBase.PaintTo
    '視点をユーザのキャラクターに合わせる
```

```
    directXPanel.GetDirect3DDevice.Transform.View =
Matrix.LookAtLH(myPlayer_.ViewPosition, myPlayer_.ViewTarget, New Vector3(0.0F,
1.0F, 0.0F))
```

下記はRefreshメソッドで画面を更新しています。

```
    '再描画を通知
    directXPanel.Refresh()
  End Sub

  Private Sub TimeUp()
    ' GameOverSceneへ描画の切り替え
    main.GameOver()
  End Sub
```

下記は自分のプレイヤー速度です。内容は先述した２Dレースゲームと全く同じです。

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Debug.WriteLine("button click")
    myPlayer_.SpeedUp(0.1)
  End Sub
End Class
```

・ ゲームオーバーシーン

誰か一人がゴールした瞬間にScene_インスタンスはGameOverシーンに以降します。このシーンでは、３DCGオブジェクトそのままにカメラが座標（0,0, -52）を中心に回ります。

下記はソースコードです。（抜粋）

```
  ' ゴールの周りをカメラが回る演出
  Private timerCounter_ As Single = 0.0F
  Public Sub SceneBase_PaintTo() Implements SceneBase.PaintTo
    Dim matrix As Matrix
    Dim viewPosition As Vector3
    Dim viewTarget As Vector3
    viewTarget = New Vector3(0, 0, -52)
```

下線部分が回転処理になります。CosやSinを使っていますが、自己の能力的にも、本作品は二次元的な回転なのであまり難しい計算ではありません。

```
    viewPosition = New Vector3(6 * Math.Cos(timerCounter_), 3, -52 + 6 * Math.Sin(timerCounter_))

    directXPanel.GetDirect3DDevice.Transform.View = matrix.LookAtLH(viewPosition, viewTarget, New Vector3(0.0F, 1.0F, 0.0F))
    timerCounter_ += 0.1F
    directXPanel.Refresh()
  End Sub
```

以上で３Dレースゲームの解説は終わりとなります。非常に多くのクラスを必要としましたが、３DCGを用いたためであり、想像よりはわかりやすく製作することができました。

三章　P2Pチャット製作

・　チャット製作について

　私はオンラインゲーム製作にあたり、ネットワーク構築の学習をするためにP2Pモデルのチャットソフトを作成いたしました。本作品は冒頭で述べたとおり、APIのDirectPlayを利用したものです。

・　ネットワークモデルについて

 DirectPlayでは、ネットワークモデルとして、サーバとクライアントからなる「クライアント/サーバモデル」と、一つのゲームに対して参加するコンピューターがお互いを接続、通信する方式の「P2P（ピアツーピア）モデル」の二種類のネットワーク形態をサポートしており、個人では前者のような費用のかかることはできないので、当然後者を用いて製作いたしました。しかし、サーバを用いないので、負荷が分散するなどといったメリットもありますが、P2Pモデルには一度に２０～３０人までが限界というデメリットもあります。ちなみにDirectPlayでは一つのゲームに参加しているプレイヤーの集まりを「セッション」と呼びます。

・　チャットソフトについて

　この作品はよりわかりやすくするために、ChatHostとChatPeerの二つのプロジェクトよりできています。

ChatHost

クラス名	説明
PeerManager	P2P通信を制御するクラス
PlayerList	複数のプレイヤーを管理するクラス
ChatHost	チャットホスト全体を制御するクラス
ChatHostForm	チャットホストのウィンドウフォーム

ChatPeer

クラス名	説明
ChatPeer	チャットアプリケーション全体を制御するクラス
ChatPeerForm	チャットアプリケーションのウィンドウフォーム

各クラスの主な役割

1、Chathost…ChatPeerのセッションを作成する。
2、ChatHostForm…アプリケーションの起動、ログの表示
3、ChatPeer…Peer間のメッセージの送受信
4、PeerManager…サービスプロバイダをTCP/IPに設定する。指定ホストに接続してピアセッションを作成し、ホストを検索する。

・PeerManager について

チャットアプリケーション作成の前に PeerManager について説明いたします。PeerManager というのは Chathost ではセッションを作成する処理を行い、ChatPeer ではホストを検索し、セッションに参加する処理を行います。

PeerManager のプロパティ

プロパティ	型	説明
ServiceProvider	Guid	サービスプロバイダ（デフォルトは TCP/IP）
UserName	String	ユーザ名
ApplicationGuid	Guid	アプリケーションの GUID
ApplicationName	String	アプリケーションの名前
IsHost	Boolean	ホストかどうか示すフラグ

まず、どのサービスプロバイダを利用するかデバイスアドレスに設定しなければいけないので、デバイスアドレスを TCP/IP に設定します。

ソースコード（抜粋）

```
Public Sub New(ByVal peer As Peer, ByVal application As Guid, ByVal applicationName
As String)
    peer_ = peer
    applicationGuid_ = application
    applicationName_ = applicationName

    'サービスプロバイダをTCP/IPに設定する
    serviceProvider_ = Address.ServiceProviderTcpIp
  End Sub

  'セッションを作る
  Public Sub CreateSession(ByVal playerName As String, ByVal sessionName As String)
    Dim appDescription As ApplicationDescription

    'DeviceAddressを設定します。
    ' 使用するプロトコルさえ知っていればよいので、ServiceProviderだけを設定します。
    Dim deviceAddress As Address = New Address()
    deviceAddress.ServiceProvider = ServiceProvider
```

次にアプリケーションの説明を ApplicayinDescription に設定します。
ここで設定する Guid には VB.net のツールにある guiden.exe で取得できる Guid を使います。
Flags というのはアプリケーションの GUID、任意の文字列のセッション名を設定し、セッションがホストの移行をできるようにするものです。

```
' アプリケーションの動作環境をApplicationDescription構造体に設定します。
    appDescription = New ApplicationDescription()
    appDescription.GuidApplication = Me.ApplicationGuid
    appDescription.SessionName = sessionName
    appDescription.Flags = SessionFlags.MigrateHost
```

次にプレイヤー情報を PlayerInfomation 構造体に設定し、Peer オブジェクトの SetPeerInfomation メソッドを呼び出してプレイヤー情報を登録します。第二引数には P2P なので SyncFlags.PeerInfomation を設定しています。

```
' Peerオブジェクトに、プレイヤー情報を設定します。
    SetPlayerInfomation(playerName)
  ' Peerオブジェクトに、プレイヤー情報を設定します。
  Public Sub SetPlayerInfomation(ByVal playerName As String)
    playerName_ = playerName
    Dim playerInformation As New PlayerInformation()
    playerInformation.Name = playerName
    peer_.SetPeerInformation(playerInformation, SyncFlags.PeerInformation)
  End Sub
```

最後に、Host メソッドを呼び出してセッションを作成します。

```
' このPCを、ピアホストとして起動します。
    Me.peer_.Host(appDescription, deviceAddress,
HostFlags.OkToQueryForAddressing)
    isHost_ = True
  End Sub
```

以上がホストアプリケーションを作るための必要な部分です。以下は作成したセッションに参加するための機能です。

ホストをまず検索してアプリケーションは始めて機能するので、Peer オブジェクトの FindHosts メソッドを実行させ、ホストが見つかると、Peer.FindHostResponse イベントハンドラが呼び出されます。イベントメッセージには、見つかったアプリケーションの設定（ApplicationDescription）や、ホストのアドレス、デバイスアドレスが保持されます。

```
' セッションの検索を開始する。検索結果はPeer.FindHostsイベントハンドラへ通知される
  Public Sub FindHostsStart()
    'Time to enum our hosts
    Dim appDescription As New ApplicationDescription()
    appDescription.GuidApplication = Me.ApplicationGuid

    Dim deviceAddress As Address = New Address()
    deviceAddress.ServiceProvider = ServiceProvider

    peer_.FindHosts(appDescription, deviceAddress, deviceAddress, Nothing, 10, 0,
```

```
0, FindHostsFlags.OkToQueryForAddressing)
  End Sub
```

また、セッションに参加するには、Peerオブジェクトの Connectメソッドを呼び出しています。引数には、先述したアプリケーションの説明と、ホストのアドレス、デバイスアドレスが最低限必要です。

```
' 指定したアドレスに接続を試みる
  Public Sub Connect(ByVal appDescription As ApplicationDescription, ByVal
hostAddress As Address, ByVal deviceAddress As Address)
    peer_.Connect(appDescription, hostAddress, deviceAddress, Nothing,
ConnectFlags.OkToQueryForAddressing)
  End Sub
End Class
```

・ ホストアプリケーションの作成

ChatHostForm の役割は非常にシンプルなもので、ログメッセージを表示するだけです。また、起動したときに ChatHost オブジェクトを初期化します。

ソースコード（抜粋）

```
  Private chatHost_ As ChatHost

  Private Sub ChatLobby_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    AppendTextLine("ピアホストを起動しています")
    chatHost_ = New ChatHost(Me)
    AppendTextLine("ピアホストが起動されました")
  End Sub

  Public Sub AppendTextLine(ByVal text As String)
    LogTextBox.AppendText(text)
    LogTextBox.AppendText(ControlChars.CrLf)
  End Sub
```

以上が ChatHostForm となります。また、以下に述べる Chathost クラスは上記の ChatHostForm オブジェクトと PeerManager オブジェクトを制御します。

ソースコード（抜粋）

```
Public Sub New(ByRef chatHostForm As ChatHostForm)
    chatHostForm_ = chatHostForm
    InitializeDirectPlay()
  End Sub
```

DirectPlayの初期化です

```
  Public Sub InitializeDirectPlay()
    peer_ = New Peer()
```

まず、Peerクラスに用意されているイベントハンドラの登録を行います。その際、最低限必要なもののみ登録します。

```
    ' イベントハンドラの登録（必要最低限のイベントのみ）
    AddHandler peer_.PlayerCreated, AddressOf Me.PlayerCreated
    AddHandler peer_.PlayerDestroyed, AddressOf Me.PlayerDestroyed
    AddHandler peer_.Receive, AddressOf Me.DataReceived
    AddHandler peer_.SessionTerminated, AddressOf Me.SessionTerminated
```

次に、PeerManagerオブジェクトの初期化を行い、その際にはPeerオブジェクトと、チャットアプリケーションのGUID、アプリケーション（ChatPeer）の名前が必要になります。その後、CreateSessionメソッドを呼び出して、新しいピア・セッションを作成します。

```
    ' PeerManagerの初期化、ピア･セッションの作成
    peerManager_ = New PeerManager(peer_, applicationGuid_, "Chat Peer")
    peerManager_.CreateSession("HOST", "ChatPeer session")
```

そして DirectPlay の初期化が終われば、参加プレイヤーを待つ状態になり、参加プレイヤーがきた場合はプレイヤー一覧を管理する PlayerList オブジェクトに、新しいプレイヤー情報が追加されます。しかし、この時に注意すべきは PlayerCreate イベントが同時に複数発生するかもしれないのです。同時に発生し、書き込もうとすると、最悪データが破壊される可能性があるのです。そのため、下記のソースにあるように Synclock ステートメントを使います。そうすることで、Synclock～End 間の実行中の処理があるときは終わるまで待つように処理されます。

ソースコード（抜粋）

```
' プレイヤーが生成された（セッションに新規参加した）時の処理
  Private Sub PlayerCreated(ByVal sender As Object, ByVal e As
PlayerCreatedEventArgs)
    ' プレイヤー情報(PlayerInformation)を取得する。
    Dim peerInfo As PlayerInformation =
peer_.GetPeerInformation(e.Message.PlayerID)
    Dim player As New Player(e.Message.PlayerID, peerInfo.Name)

    ' プレイヤー情報をplayerList_に追加する。
    ' マルチスレッドで動作するため、データが破壊されないように同期を取る
    SyncLock playerList_
      playerList_.Add(player)
    End SyncLock
  End Sub
```

プレイヤーが退場したときも同様です。

```
  ' プレイヤーが何らかの理由によりセッションから削除されたときの処理
  Private Sub PlayerDestroyed(ByVal sender As Object, ByVal e As
PlayerDestroyedEventArgs)
    ' playerList_から削除する
    SyncLock playerList_
      Dim player As Player
      For Each player In playerList_
        If e.Message.PlayerID = player.PlayerID Then
          playerList_.Remove(player)
          Exit For
        End If
      Next player
    End SyncLock
  End Sub
```

次に PlayerList クラスです。３D レースの時と同じように ArrayList クラスを継承したク

ラスです。複数の Player 構造体を管理しており、PlayerID から名前を検索する GetPlayerName メソッドを実装しています。また、このクラスも実行中に他のイベントの処理で修正される可能性があるので、SyncLock ステートメントで同期を取っています。

ソースコード

```
Public Class PlayerList
  Inherits System.Collections.ArrayList
  ' 指定のプレイヤーIDから、playerList_内を検索してプレイヤー名を返す
  Public Function GetPlayerName(ByVal idPlayer As Integer) As String
    SyncLock Me
      Dim player As Player
      For Each player In Me
        If player.PlayerID = idPlayer Then
          Return player.Name
        End If
      Next player
    End SyncLock
    Return Nothing
  End Function 'GetPlayerName
End Class

' プレイヤーの情報をまとめる構造体
Public Structure Player
  Public PlayerID As Integer
  Public Name As String

  Public Sub New(ByVal playerID As Integer, ByVal name As String)
    Me.PlayerID = playerID
    Me.Name = name
  End Sub
End Structure
```

以上でホストアプリケーションの解説は終了です。

・ チャットアプリケーションについて

・

次にアプリケーション本体の作成です。ChatPeerForm の役割は、ChatPeer オブジェクトの指示に従って、メッセージを ChatTextBox に表示し、SendMsgTextBox に入力されたテキストを Chatpeer オブジェクトに送信することです。

ChatPeerForm は、まず ChatPeer が起動したときに、後述する通信を制御する ChatPeer オブジェクトを初期化します。

ソースコード（抜粋）

```
Private Sub ChatPeerForm_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    chatPeer_ = New ChatPeer(Me)
  End Sub
```

次に、送信ボタンが押された時に SendMsgTextBox に入力された文字列をセッション参加中の全プレイヤーに送信する事を ChatPeer オブジェクトの SendMessage メソッドで依頼しています。

ソースコード（抜粋）

```
' 送信ボタンが押されたらメッセージを送信
  Private Sub SendButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SendButton.Click
    chatPeer_.SendMessage(SendMsgTextBox.Text)
    SendMsgTextBox.Text = Nothing
  End Sub
```

また、下記の AppendTextLine メソッドは ChatTextBox に引数の文字列を表示します。この処理はメッセージ送信イベントが受信される事によって ChatTextBox の表示内容が壊されないように SyncLock ステートメントで同期を取っています。

```
' チャットメッセージの追加
  Public Sub AppendTextLine(ByVal message As String)
    SyncLock ChatTextBox
      ' テキストボックスの最大文字数に近づいたら古いメッセージを削除
      If ChatTextBox.Text.Length > ChatTextBox.MaxLength * 0.95 Then
        ChatTextBox.Text = ChatTextBox.Text.Remove(0, CInt(ChatTextBox.MaxLength /
2))
```

```
      End If
      ' メッセージの追加
      ChatTextBox.AppendText(message)
      ChatTextBox.AppendText(ControlChars.CrLf)
      ChatTextBox.SelectionStart = ChatTextBox.Text.Length
      ChatTextBox.ScrollToCaret()
    End SyncLock
  End Sub
```

そして最後に終了処理になります。

```
  ' Form は dispose をオーバーライドしてコンポーネント一覧を消去します。
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    ' Peerセッションは、コントロールが破棄される前にDisposeします。
    If Not (chatPeer_ Is Nothing) Then
      chatPeer_.Dispose()
    End If

    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)

    Application.Exit()
  End Sub
```

次にChatPeerクラスの解説に移ります。GUIとメッセージの送受信の制御を行うもので、つまりはChatHostクラスをベースに、クライアントとしての機能を追加したものと考えてください。
また、新しくイベントハンドラを登録しています。

イベント	説明
FindHostResponse	FindHostsの呼び出しにホストが応答すると発生する。

HostMigrated　　ホストが移行すると発生する。

ソースコード（抜粋）

```
' イベントハンドラの登録（必要最低限のイベントのみ）
    AddHandler peer_.PlayerCreated, AddressOf Me.PlayerCreated
    AddHandler peer_.PlayerDestroyed, AddressOf Me.PlayerDestroyed
    AddHandler peer_.Receive, AddressOf Me.DataReceived
    AddHandler peer_.SessionTerminated, AddressOf Me.SessionTerminated
    AddHandler peer_.FindHostResponse, AddressOf Me.FindHostResponsed
    AddHandler peer_.HostMigrated, AddressOf Me.HostMigrated
```

また、FindHostResponsedイベントは、ホストとなるアプリケーション（ChatHost）が見つかったときに呼び出されるイベントです。ここでChathostでは次にセッションを作る処理をしましたが、ChatPeerはホストではないので、PeerManagerオブジェクトのFindHostStartメソッドでホストの検索を開始し、セッションでの参加処理を始めます。

```
'ホストの検索を開始する。ホストが見つかるとFindHostResponseイベントハンドラが呼び出される。
    chatPeerForm_.AppendTextLine("ホストの検索中...")
    peerManager_.FindHostsStart()
  End Sub
' セッションが検索できたら呼ばれるイベントハンドラ
  Private Sub FindHostResponsed(ByVal sender As Object, ByVal e As
Microsoft.DirectX.DirectPlay.FindHostResponseEventArgs)
    chatPeerForm_.AppendTextLine("ホストが見つかりました。")
```

また、次のConnectメソッドを実行する前に、まず自分が誰なのかをPeerオブジェクトに知らせる必要があり、ここでは名前に乱数を使い、設定しています。

```
    ' プレイヤー情報(PlayerInfomation)の設定
    chatPeerForm_.AppendTextLine("ホストに接続しています...")
    Dim random As Random = New Random()
    Dim username As String = "player" & CInt(random.NextDouble * 100)
    peerManager_.SetPlayerInfomation(username)
```

```
    ' セッションへの接続
    Try
```

Connectメソッドの引数にはFindHostResponsedイベントハンドラの引数FindHostResponseEventArgsに保管されているアプリケーションの説明e.Message.ApplicationDescriptionと、デバイスアドレスe.Message.AddressDeviceを設定します。

```
      peerManager_.Connect(e.Message.ApplicationDescription,
e.Message.AddressSender, e.Message.AddressDevice)
    Catch ex As Exception
      chatPeerForm_.AppendTextLine("セッションへの接続に失敗しました。")
      chatPeerForm_.AppendTextLine(ex.Message & ex.StackTrace)
    End Try
    ' ホストが見つかったので、これ以上のホストの検索を終了する。
    peer_.CancelAsyncOperation(CancelFlags.FindHosts)
    RemoveHandler peer_.FindHostResponse, AddressOf Me.FindHostResponsed

    chatPeerForm_.AppendTextLine("ホストに接続が完了しました。あなたの名前は" &
username & "です。")
  End Sub
```

また、段落の冒頭で述べたHostMigratedイベントハンドラがいつ使われているのかを説明します。

```
  ' ホストがmigrated（今までのホストPCからセッション内の別のPCに移動）したときの
処理
  Private Sub HostMigrated(ByVal sender As Object, ByVal e As HostMigratedEventArgs)
    Debug.WriteLine("host migrated")
    'もし自分がホストに選ばれたらウィンドウのタイトルに文字列(HOST)を追加する。
    If localPlayerId_ = e.Message.NewHostID Then
      chatPeerForm_.Text += "(HOST)"
    End If
  End Sub
```

上記ではHostMigrateイベントハンドラには、自分自身が新しいホストに選ばれた場合の処

理を記述しています。また、ホストに選ばれたプレイヤーのチャットアプリケーションのタイトルバーに（HOST）の文字列を追加する処理が記述されています。

　プレイヤーが新規に参加した場合はChatHostと同様ですが、もし追加されたプレイヤーが自分自身だった場合には、下記のlocalPlayerId_変数にプレイヤーIDを保持します。

```
' プレイヤーが生成された（セッションに新規参加した）時の処理
  Private Sub PlayerCreated(ByVal sender As Object, ByVal e As
PlayerCreatedEventArgs)
    ' プレイヤー情報(PlayerInformation)を取得する。
    Dim peerInfo As PlayerInformation =
peer_.GetPeerInformation(e.Message.PlayerID)
    Dim player As New Player(e.Message.PlayerID, peerInfo.Name)

    ' プレイヤー情報をplayerList_に追加する。
    ' マルチスレッドで動作するため、データが破壊されないように同期を取る
    SyncLock playerList_
      playerList_.Add(player)
    End SyncLock
    chatPeerForm_.AppendTextLine(player.Name & "さんが参加されました。")

    ' もし自分自身だったらlocalPlayerId_にプレイヤーIDを保存する。
    If peerInfo.Local Then
      localPlayerId_ = e.Message.PlayerID
    End If
```

プレイヤーが退場したときの処理はChatHostと同様です。ただし、退場を知らせるメッセージを追加しました。

```
' プレイヤーが何らかの理由によりセッションから削除されたときの処理
  Private Sub PlayerDestroyed(ByVal sender As Object, ByVal e As
PlayerDestroyedEventArgs)
    ' playerList_から削除する
    SyncLock playerList_
      Dim player As Player
      For Each player In playerList_
        If e.Message.PlayerID = player.PlayerID Then
          playerList_.Remove(player)
```

```
          Exit For
        End If
      Next player
      chatPeerForm_.AppendTextLine(player.Name & "さんが退場しました。")
    End SyncLock
```

最後に、メッセージの送受信に関してです。まず、送信の場合にはPeerオブジェクトのSendToメソッドを使います。送信するメッセージとなるネットワークパケットを作成し、SendToメソッドの引数に「送信先のプレイヤーID　　（全員に送る場合はCInt<PlayerID.AllPlayers>」、「ネットワークパケット」、「タイムアウトの時間（通常は０）」、「送信方法のオプションとなるフラグ」を設定し、SendToメソッドを呼び出します。ここではユーザーが入力したメッセージ（Message）を、全員に（Cint＜PlayerID.Allplaers＞）メッセージが確実に届くように（SendFlags.Garanteed）送信しています。

```
' 送信ボタンが押されたらメッセージを送信
  Public Sub SendMessage(ByVal message As String)
    Dim data As New NetworkPacket()
    ' ネットワークパケットにメッセージを書き込む
    data.Write(message)

    peer_.SendTo(CInt(PlayerID.AllPlayers), data, 0, SendFlags.Guaranteed)
    End Sub
```

一方、プレイヤーがメッセージを受信すると、PeerオブジェクトのReceiveイベントを登録したChatPeerのDataRecievedイベントハンドラが呼び出されます。受信したメッセージはイベントハンドラの引数RecieveEventArgsのMessageプロパティ内部に保管されています。Messageプロパティは、RecieveMessage構造体で、メッセージの内容のほかに、そのメッセージを送信したプレイヤーのIDなどが保管されています。

```
' メッセージを受信したときの処理
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
    ' ネットワークパケットを文字列として受信
    Dim message As String = e.Message.ReceiveData.ReadString()
    ' メッセージを書き込む()
    chatPeerForm_.AppendTextLine(playerList_.GetPlayerName(e.Message.SenderID) &
"> " & message)
```

```
    e.Message.ReceiveData.Dispose() ' 長期間メッセージを保持する必要が無ければ破棄する
  End Sub
```

以上でP2Pチャットに関する解説は終わりです。

四章　オンライン３Dレースゲームについて

いよいよこの論文の本題に入ります。この章では、今までの章の事全てを踏まえたうえで解説していきます。まず、プロジェクトはPeerChatと同じく二つに分かれ、クラスが派生しています。

RaceGameOnlineHost

クラス名	説明
PeerManager	P2P通信を制御するクラス
PlayerList	複数のプレイヤーを管理するクラス
RaceGameOnlineHost	ホスト全体を制御するクラス
RaceGameOnlineHostForm	ホストのウィンドウフォーム

RaceGameOnline

クラス名	説明
DirectXPanel	DirectXの表示を管理するクラス
GameTimer	ゲーム内の時間を管理するクラス
GameDataTime	ピア・セッション内の時間を同期するためのクラス
GamePeer	レースゲームの通信部分を制御するクラス
LoginForm	ログイン画面のウィンドウフォーム
MeshObject	メッシュの形状データ、マテリアル、テクスチャを管理するクラス
MeshObjectList	複数のMeshObjectオブジェクトを管理するクラス
Message	メッセージを判別し、メソッドを呼び出す機能
SceneBase	シーンの基底となるインターフェイス
Player	プレイヤー情報を管理するクラス
PlayerList	複数のPlayerオブジェクトを管理するクラス
CountDownScene	カウントダウンとゲームの初期化を行うシーンクラス
GameMainScene	レースゲーム本体のシーンクラス

GameTitleScene	タイトル画面のシーンクラス
GameOverScene	ゲームオーバー画面のシーンクラス
RaceForm	レースゲーム全体を管理するクラス

・ プロトコルの設計

ここでは受け取ったメッセージを判別して、適切な処理を行うメソッドを呼び出す機能を持つMessageクラスを解説します。

なぜこの機能が必要かというのは、チャットアプリケーションとは違い、Peer同士がやり取りする情報がメッセージだけではなく、座標データ、Playerの詳細情報、チャットメッセージなどの様々な情報を扱い、その際にReceiveイベントハンドラがどんなものでも受け取ってしまうためです。

そのため、ここではやり取りする情報の先頭に、メッセージの種類を現す値、メッセージIDを付加するようにして、Receiveイベントハンドラがメッセージに従って各種の処理を行うようにしています。このメッセージIDは、MessageType列挙型を使って表現しています。

MessageType列挙型

メッセージタイプ	説明
PositionMessage	座標データをやり取りするときに使うメッセージ
PlayerStatusMessage	プレイヤーの名前やプレイヤーIDなど、プレイヤーに関する詳細な情報をやり取りするときに使うメッセージ
GameStartMessage	ゲームの開始を通知するメッセージ
SyncTimeMessage	ピア・セッション内で時間を合わせるためのメッセージ

```
' メッセージの種類を表す列挙型
Public Enum MessageType
  PositionMessage
  PlayerStatusMessage
  GameStartMessage
  SyncTimeMessage
End Enum
```

以上のメッセージの中身をMessageクラスとして定義します。しかし、ピア・セッション内でやり取りできるデータはバイト配列なので、メッセージをバイト配列に変換する仕組みが必要となります。

Messageクラス

メソッド名	引数	戻り値	説明
EncodeBytes	MessageAsMessage	Byte（）	メッセージをバイト配列にエンコードする
EncodeBytes	binaryWriterAsBinaryWriter		指定したストリームにエンコードする情報を書き込む
DecodeBytes	bufferAsByte()	Message	指定されたバイト配列からメッセージにデコードする

このメッセージを基に座標データをやり取りするPositionMessageクラスや、プレイヤーの情報をやり取りするPlayerStatusMessageクラスなどのサブクラスが作られます。
次の二つの段落では例として座標データのエンコード/デコードを順に解説します。

・ エンコード

座標データをピア・セッション内のプレイヤーに送信するときは、まずPositionMessageオブジェクトに自分の座標データを保管し、そのままではまだVector3構造体のため送信できないので、Vector3クラスの情報をバイト配列に変換（エンコード）してあげる必要があります。そのため、それを行うEncodeBytesメソッドを作ります。

この時、様々な型の情報をバイト配列に変換するにはネットワーク処理にも利用できるストリームを利用するのが便利であり、ストリームに書き込んだ内容をバイト配列で取得できる機能を持ったMemoryStreamクラスのインスタンス、Memoryを生成します。また、このMemoryStreamクラスにはストリームとして最低限の機能しか持たないため、文字列や少数など様々な型の値をストリームに書き込めるように、BinaryWriterクラスを更に用意しています。

ソースコード（抜粋）

```
' メッセージをバイト配列に変換する。実装は派生クラス。
  Public Shared Function EncodeBytes(ByRef message As Message) As Byte()
    'Debug.WriteLine("Mesage.EncodeBytes(message)")
    Dim result As Byte()
    Dim memory As MemoryStream = New MemoryStream()
    Dim binaryWriter As BinaryWriter = New BinaryWriter(memory)
```

BinaryWriterオブジェクト介して、まずメモリに書き込まれるのはメッセージIDです。勿論ここでは座標データを表すPositionMessageが書き込まれます。その後にメッセージの内容を書き込むEncodeBytes（binaryWriterAsBinaryWriter）メソッドを呼び出します。

```
    ' はじめにメッセージタイプを書き込む
    binaryWriter.Write(message.MessageType)
    ' 次に送信するデータを書き込む
    message.EncodeBytes(binaryWriter)

    ' データをバッファに確実に書き込む
    binaryWriter.Flush()

    ' メモリからbyte配列を取得する
    result = memory.GetBuffer()
    Return result
  End Function
```

次に座標データの記述ですが、下記の部分は当然、メッセージの種類によって、どのように処理をするのか変わります。座標データの書き込み処理は、Messageクラスを継承したサブクラス、PositionMessageクラスの中に記述します。Write（ValueAsSingle）メソッドを使って、Vector3構造体の各要素、ｘ座標、ｙ座標、ｚ座標のSingle値を順番に書き込みます。

```
' 座標データ
Public Class PositionMessage
  Inherits Message

  Public Sub New()
    Me.MessageType = MessageType.PositionMessage
  End Sub

  Public Position As Microsoft.DirectX.Vector3
  ' メッセージをバイト配列に変換する。
  Public Overloads Overrides Sub EncodeBytes(ByRef binaryWriter As BinaryWriter)
    'Debug.WriteLine("PositionMessage.EncodeBytes(binaryWriter)")
```

```
    binaryWriter.Write(Me.Position.X)
    binaryWriter.Write(Me.Position.Y)
    binaryWriter.Write(Me.Position.Z)
  End Sub
```

メッセージID　+ x座標 + Y座標 + z座標　　（全て３２bit）

＝１２８bit ＝１６byte

上記のようにしてバイト配列に変換された座標データは、DirectPlayのPeerObjectに用意されたSendToメソッドでプレイヤーに送信されます。

※下記はGamePeerからの抜粋

```
Dim data As New NetworkPacket()
    ' ネットワークパケットにメッセージを書き込む
    data.Write(message.EncodeBytes(message))
    ' ネットワークパケットを全員に送信
    Try
      peer_.SendTo(CInt(PlayerID.AllPlayers), data, 0, sendFlags)
    Catch ex As Microsoft.DirectX.DirectPlay.NoConnectionException
      Debug.WriteLine("セッションに参加していません。")
      MessageBox.Show("セッションに参加していません。")
    End Try
```

・ デコード

バイト配列に変換されて送られたメッセージは、DirectPlayのReceiveイベントハンドラで受信されます。

ソースコード（GamePeerより）

メッセージを受信したときの処理

```
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
    ' ネットワークパケットを文字列として受信
    Dim recieveData As Byte() = CType(e.Message.ReceiveData.Read(GetType(Byte),
e.Message.ReceiveData.Length), Byte())
```

```
    ' Messageオブジェクトにデコード
      Dim message As Message = message.DecodeBytes(recieveData)
```

そしてVector3構造体に戻すために、ここでもストリームを利用します。MemoryStreamオブジェクトを生成し、受信したバイト配列をメモリに書き込みます。書き込み終わった時点でストリームの位置は最後になるので、メモリの内容を読み込むためにSeekメソッドを使って、ストリームの位置を先頭に戻します。

ソースコード（Messageより抜粋）

```
' バイト配列のネットワークパケットをMessageクラスに復元する。実装は派生クラス。
  Public Shared Function DecodeBytes(ByVal buffer As Byte()) As Message
    'Debug.WriteLine("Mesage.DecodeBytes(buffer)")
    Dim result As Message = New Message()
    ' bufferの内容をメモリストリームに書き込む
    Dim memory As MemoryStream = New MemoryStream()
    memory.Write(buffer, 0, buffer.Length  - 1)
      memory.Seek(0, SeekOrigin.Begin)
```

そしてメッセージの先頭には必ずメッセージID書き込んでいるので、BinaryReaderオブジェクトを生成して先頭の３２bit整数を読み込みます。なお列挙型は内部でInteger（３２bit整数）として保管されているので、読み込むときにはIntegerとして読み込みます。読み込んだ整数値は、そのままMessageType列挙型に変換し、メッセージの種類に応じてSelectステートメントで処理を分岐します。上記のようにPositionMessageであった場合は、PositionMessageクラスの
DecodeBytesメソッドを呼び出して、座標データを読み込みます。

```
Public Shared Function DecodeBytes(ByVal buffer As Byte()) As Message
    'Debug.WriteLine("Mesage.DecodeBytes(buffer)")
    Dim result As Message = New Message()
    ' bufferの内容をメモリストリームに書き込む
    Dim memory As MemoryStream = New MemoryStream()
    memory.Write(buffer, 0, buffer.Length  - 1)
    memory.Seek(0, SeekOrigin.Begin)
    ' bufferの内容を読み込むReaderを生成する
```

```
    Dim binaryReader As BinaryReader = New BinaryReader(memory)

    ' はじめにメッセージタイプを読み込む
    ' Enum型は、内部的にはIntegerなのでInt32で読み込み、CTypeで変換できる
    Dim messageType As MessageType = CType(binaryReader.ReadInt32(), MessageType)

    ' メッセージタイプにしたがって、受信するデータを読み込む
    Select Case messageType
      Case messageType.PositionMessage
        Dim positionMessage As PositionMessage = New PositionMessage()
        result = positionMessage.DecodeBytes(binaryReader)
          End Select
    Return result
  End Function
```

そして最後にエンコード時と同じ順番通りに、Single値を読み込みます。

```
' 座標データ
Public Class PositionMessage
  Inherits Message

  Public Sub New()
    Me.MessageType = MessageType.PositionMessage
  End Sub

  Public Position As Microsoft.DirectX.Vector3
  ' メッセージをバイト配列に変換する。
  Public Overloads Overrides Sub EncodeBytes(ByRef binaryWriter As BinaryWriter)
    'Debug.WriteLine("PositionMessage.EncodeBytes(binaryWriter)")
    binaryWriter.Write(Me.Position.X)
    binaryWriter.Write(Me.Position.Y)
    binaryWriter.Write(Me.Position.Z)
  End Sub

  ' メッセージをバイト配列から復元する。
  Public Overloads Overrides Function DecodeBytes(ByRef binaryReader As
```

```
BinaryReader) As Message
    Dim result As PositionMessage = New PositionMessage()
    result.Position.X = binaryReader.ReadSingle()
    result.Position.Y = binaryReader.ReadSingle()
    result.Position.Z = binaryReader.ReadSingle()
    'Debug.WriteLine("PositionMessage.DecodeBytes(binaryReader):" &
result.Position.ToString())
    Return result
  End Function
```

その他のメッセージについても原理は同じなので、ここでは説明は省略させて頂きます。

・ ログイン処理について

まず、下記にLoginFormの全ソースコードがありますが、これに関しては解説は不要だと思いますので、特に解説はいたしません。

```
Public Class LoginForm
    Inherits System.Windows.Forms.Form
Private gamePeer_ As GamePeer
  Public Sub New(ByRef gamePeer As GamePeer)
    Me.New()
    gamePeer_ = gamePeer
  End Sub
  Private Sub LoginButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LoginButton.Click
    Try
      gamePeer_.Login(HostAddressTextBox.Text, PlayerNameTextBox.Text,
charactorType_)
    Catch ex As Exception
      MessageBox.Show("ログインできませんでした。存在しないホストアドレスです。")
    End Try
  End Sub
  Public Sub AppendTextLine(ByVal message As String)
    LogTextBox.AppendText(message)
    LogTextBox.AppendText(ControlChars.CrLf)
  End Sub
```

```
  Private charactorType_ As String = "red.x"
  Private Sub CharactorTypeComboBox_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CharactorTypeComboBox.SelectedIndexChanged
    Debug.WriteLine("LoginForm.Selected" & CharactorTypeComboBox.SelectedItem)
    Select Case CharactorTypeComboBox.SelectedItem
      Case "赤"
        charactorType_ = "red.x"
      Case "青"
        charactorType_ = "blue.x"
      Case "黄"
        charactorType_ = "yellow.x"
    End Select
  End Sub
End Class
```

これに関連して、ログインボタンを押すと、ユーザーが入力したアドレスに対して、ホストが存在するかどうかを検索し、PeerManagerオブジェクトのFindHostStartメソッドを呼び出してホストの検索を開始します。

ソースコード（GamePeerより抜粋）

```
Private playerName_ As String
  Private charactorType_ As String
  Public Sub Login(ByVal hostAddress As String, ByVal playerName As String, ByVal
charactorType As String)
    If playerName = "" Then
      playerName_ = "guest" ' 名前を入力していなかったらguestにする
    Else
      playerName_ = playerName
    End If
    charactorType_ = charactorType
    Debug.WriteLine("GamePeer.Login:charactorType=" & charactorType_)
    'ホストの検索を開始する。ホストが見つかるとFindHostResponseイベントハンドラが
呼び出される。
```

```
    loginForm_.AppendTextLine("ホストの検索中...")
    peerManager_.FindHostsStart(hostAddress)
  End Sub
```

チャットアプリケーションでは必要最低限の情報しか設定していません。今回はホストのアドレスを入力するGUIを用意して、FindHostStartメソッドを変更し、あらかじめホストの検索に必要な情報を揃えています。ホストアドレスの設定は、Addressクラスの AddComponentメソッドを使っています。第一引数に設定するキーの名前、第二引数に値を設定します。

ソースコード（PeerManager　　より抜粋）

```
' 指定のアドレスにホストがあるかどうかを検索する
  ' 検索結果はPeer.FindHostsイベントハンドラへ通知される
  Public Sub FindHostsStart(ByVal hostAddressString As String)
    'Time to enum our hosts
    Dim appDescription As New ApplicationDescription()
    appDescription.GuidApplication = Me.ApplicationGuid

    Dim hostAddress As Address = New Address()
    hostAddress.ServiceProvider = ServiceProvider
    hostAddress.AddComponent("hostname", hostAddressString)
    Dim deviceAddress As Address = New Address()
    deviceAddress.ServiceProvider = ServiceProvider

    peer_.FindHosts(appDescription, hostAddress, deviceAddress, Nothing, 10, 0, 0,
FindHostsFlags.OkToQueryForAddressing)
  End Sub
```

また、GamePeerクラスの話に戻りますが、コードはほとんど先述したChatPeerクラスと同じであり、下記にあるように、ホストアドレスを指定してホストの検索をする FindHostStartメソッドと、DataRecievedメソッドが大幅に修正されています。

ソースコード（GamePeerより抜粋）

```
'ホストの検索を開始する。ホストが見つかるとFindHostResponseイベントハンドラが呼び
```

出される。

```
    loginForm_.AppendTextLine("ホストの検索中...")
    peerManager_.FindHostsStart(hostAddress)
  End Sub

' メッセージを受信したときの処理
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
    ' ネットワークパケットを文字列として受信
    Dim recieveData As Byte() = CType(e.Message.ReceiveData.Read(GetType(Byte),
e.Message.ReceiveData.Length), Byte())
```

・ タイトル処理

次にタイトルシーンの説明です。GameTitleSceneはPlayerListに更新があったかどうかを定期的にチェックして、更新があった場合にPlayerListTextBoxの表示を更新します。また、プレイヤーがスタートボタンを押すと、RaceFormオブジェクトのGameStartメソッドを呼び出して、他のプレイヤーにゲームの開始を通知します。

ソースコードの解説としてまず、RaceFormオブジェクトのGameStartableプロパティがTrueになっている場合、OnGameStartメソッドを呼び出してゲームを開始します。GameStartableプロパティは他のプレイヤーがスタートボタンを押した時に設定されるプロパティです。

全ソースコード（GameTitleScene）

```
Public Class GameTitleScene
  Inherits System.Windows.Forms.UserControl
  Implements SceneBase
Private main As RaceForm
  Public Sub New(ByRef main As RaceForm)
    MyBase.New()
    ' この呼び出しは Windows フォーム デザイナで必要です。
    InitializeComponent()
    Me.main = main
  End Sub

  Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartButton.Click
```

```
    main.GameStart()
  End Sub
```

以下の下線部が解説した部分になります。

```
  Private Sub SceneBase_MoveTo() Implements SceneBase.MoveTo
    If main.GameStartable = True Then
      main.GameStartable = False
      main.OnGameStart()
    End If
  End Sub

  Private playerCount_ As Integer = 0
  Private Sub SceneBase_PaintTo() Implements SceneBase.PaintTo
    'Debug.WriteLine("GameTitleScene.PaintTo")
    'PlayerListが更新されているかチェック
    If main.PlayerList.Count <= playerCount_ Then Return

    ' 現在のプレイヤーの数を保管する
    playerCount_ = main.PlayerList.Count

    ' プレイヤーリストをクリア
    PlayerListTextBox.Text = ""
    'PlayerListの内容を更新する
    Debug.WriteLine(" count=" & main.PlayerList.Count)
    Dim i As Integer
    SyncLock main.PlayerList

      For i = 0 To main.PlayerList.Count  - 1
        Dim name As String = main.PlayerList(i).Name
        If Not name = "HOST" Then ' ホスト以外のプレイヤーを列挙
          PlayerListTextBox.AppendText(name)
          PlayerListTextBox.AppendText(ControlChars.CrLf)
          Debug.WriteLine(" " & name)
        End If
      Next
```

```
    End SyncLock

  End Sub
End Class
```

先述したスタートボタンを押すと、ピア・セッション内の全てのプレイヤーに、これからゲームを開始する事が通達されます。このときに送信されるGameStartMessageには、ゲームをスタートする時間を示すDateTimeオブジェクト、スタート地点の各プレイヤーの座標情報を示すハッシュテーブルが含まれます。

GameStartMessage

プロパティ	型	説明
StartTime	DateTime	ゲームをスタートする時間
Positions_	HashTable	スタート地点の各プレイヤーの座標

ソースコード（RaceFormより抜粋）

```
Public Sub GameStart()
    Debug.WriteLine("RaceForm.GameStart()")

    ' 全てのプレイヤーのスタート地点の座標を設定する
    Dim positions As Hashtable = New Hashtable()
    Dim i As Integer
    For i = 0 To PlayerList.Count - 1
      positions(PlayerList.GetPlayer(i).PlayerID) = New Vector3(((i * 6) /
(PlayerList.Count - 1)) - 3, 1, 0)
    Next
' 10秒後にゲームを開始することを通知する
    Dim startTime As DateTime = DateTime.Now.AddSeconds(10)
    'Debug.WriteLine("RaceForm.GameStart:starttime " & startTime)
    'Debug.WriteLine("RaceForm.GameStart:now " & DateTime.Now)
    gamePeer_.SendMessage(New GameStartMessage(startTime, positions))

  End Sub
```

ゲーム開始の通知を受けたプレイヤーは、RaceFormオブジェクトのGameStartableプロパティをTrueに設定し、GameTitleSceneオブジェクトにゲームの開始が可能な事を検出できる

ようにしています。その後、指定の時間にゲームを開始するように、各種設定を行うRaceFormオブジェクトのSetGameStartSettingsメソッドを呼び出します。

ソースコード（GamePeerより抜粋）

```
' メッセージを受信したときの処理
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
    ' ネットワークパケットを文字列として受信
    Dim recieveData As Byte() = CType(e.Message.ReceiveData.Read(GetType(Byte),
e.Message.ReceiveData.Length), Byte())

    ' Messageオブジェクトにデコード
    Dim message As Message = message.DecodeBytes(recieveData)

    Select Case message.MessageType
      Case MessageType.PositionMessage
        ' 座標データを更新する
        main.UpdatePosition(e.Message.SenderID, CType(message, PositionMessage))
      Case MessageType.PlayerStatusMessage
        ' プレイヤー情報を更新する
        main.UpdatePlayerStatus(e.Message.SenderID, CType(message,
PlayerStatusMessage))
      Case MessageType.GameStartMessage
        ' ゲームスタートが可能なことをRaceFormに通知する
        main.GameStartable = True
        main.StartTime = CType(message, GameStartMessage).StartTime
        main.SetGameStartSettings(CType(message, GameStartMessage))
```

ゲームの開始を受けたGameTitleSceneは、RaceFormオブジェクトのOnGameStartメソッドを呼び出して、次のシーンへの切り替えを行います。

ソースコード（RaceFormより抜粋）

```
Public Sub OnGameStart()
    Debug.WriteLine("RaceForm.GameStart(gamestartmessage)")
```

```
    ' タイマーの初期化、以前のタイマーの停止
    gameTimer_.Stop()

    ' タイトル画面をはずす
    If Me.Controls.Contains(gameTitleScene) = True Then
      ' Disposeする前に、DirectXPanelをRemoveして、いっしょにDisposesされないようにする。
      gameTitleScene.Controls.Remove(directXPanel_)
      Me.Controls.Remove(gameTitleScene)
      gameTitleScene.Dispose()
    End If

    ' カウントダウンのシーンに切り替え。
    countDownScene = New CountDownScene(Me, directXPanel_)
    Me.Controls.Add(countDownScene)

    scene_ = countDownScene

    ' プレイヤー情報を送信してキャラクタータイプを更新する
    Dim playerStatusMessage As PlayerStatusMessage = New PlayerStatusMessage()
    playerStatusMessage.Player = MyPlayer
    gamePeer_.SendMessage(playerStatusMessage)

    ' タイマーのスタート
    gameTimer_.Interval = 100
    gameTimer_.Start()
  End Sub
```

・ 時間の同期

当然の事ですが、このゲームではリアルタイム対戦を行うので、時間がずれてしまうようなことがあってはなりません。行動タイミングがずれてしまいます。そこでスタートボタンを押したプレイヤーに他のプレイヤーの時間を合わせるようにします。時間の同期を要求するメッセージは、SendFlag.Sync送信オプションに加えて、相手の受信が完了するまで待ちます。

ソースコード（RaceFormより抜粋）

```
Public Sub GameStart()
    Debug.WriteLine("RaceForm.GameStart()")

    ' 全てのプレイヤーのスタート地点の座標を設定する
    Dim positions As Hashtable = New Hashtable()
    Dim i As Integer
    For i = 0 To PlayerList.Count - 1
      positions(PlayerList.GetPlayer(i).PlayerID) = New Vector3(((i * 6) /
(PlayerList.Count - 1)) - 3, 1, 0)
    Next

    ' 時間を合わせるように通知する
    Dim syncTimeMessage As SyncTimeMessage = New SyncTimeMessage()
    syncTimeMessage.Now = DateTime.Now
    gamePeer_.SendMessage(syncTimeMessage, SendFlags.Guaranteed Or SendFlags.Sync)

    ' 10秒後にゲームを開始することを通知する
    Dim startTime As DateTime = DateTime.Now.AddSeconds(10)
    'Debug.WriteLine("RaceForm.GameStart:starttime " & startTime)
    'Debug.WriteLine("RaceForm.GameStart:now " & DateTime.Now)
    gamePeer_.SendMessage(New GameStartMessage(startTime, positions))

  End Sub
```

SyncTimeMessageを受信すると、RaceFormオブジェクトのUpdateGameTimeメソッドを呼び出します。

ソースコード（GamePeerより抜粋）

```
' メッセージを受信したときの処理
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
    ' ネットワークパケットを文字列として受信
    Dim recieveData As Byte() = CType(e.Message.ReceiveData.Read(GetType(Byte),
e.Message.ReceiveData.Length), Byte())
```

```
    ' Messageオブジェクトにデコード
    Dim message As Message = message.DecodeBytes(recieveData)

    Select Case message.MessageType
      Case MessageType.PositionMessage
        ' 座標データを更新する
        main.UpdatePosition(e.Message.SenderID, CType(message, PositionMessage))
      Case MessageType.PlayerStatusMessage
        ' プレイヤー情報を更新する
        main.UpdatePlayerStatus(e.Message.SenderID, CType(message,
PlayerStatusMessage))
      Case MessageType.GameStartMessage
        ' ゲームスタートが可能なことをRaceFormに通知する
        main.GameStartable = True
        main.StartTime = CType(message, GameStartMessage).StartTime
        main.SetGameStartSettings(CType(message, GameStartMessage))
      Case MessageType.SyncTimeMessage
        main.UpdateGameTime(CType(message, SyncTimeMessage))
    End Select

    e.Message.ReceiveData.Dispose() ' 長期間メッセージを保持する必要が無ければ破
棄する
  End Sub
```

UpdateGameTimeメソッド内では、プレイヤー間の時間差を埋めるGameDateTimeオブジェクトを生成します。

ソースコード（RaceFormより抜粋）

```
' ピア・セッション内の時間の同期
  Public Sub UpdateGameTime(ByVal syncTimeMessage As SyncTimeMessage)
    Debug.WriteLine("RaceForm.UpdateGameTime")
    gameDateTime_ = New GameDateTime(syncTimeMessage.Now)
  End Sub
```

GameDateTimeクラスは、Nowプロパティを読み込むと時間差を計算して、セッション内の全てのプレイヤーが同じ時間を得られるようにします。

```
Public Class GameDateTime

  Private gap_ As Long
  Public Sub New(ByVal baseDateTime As DateTime)
    gap_ = DateTime.Now.ToFileTime() - baseDateTime.ToFileTime()
    'Debug.WriteLine("GameDateTime:gap=" & gap_)
  End Sub

  Public Function Now() As DateTime
    Return DateTime.FromFileTime(DateTime.Now.ToFileTime() - gap_)
  End Function
End Class
```

このようにして、ピア・セッション内での時間差を埋めています。

・ カウントダウンシーン

ゲームがスタートするとまずはじめにカウントダウンシーンに移ります。ここではカウントダウンの表示とプレイヤーの初期化を行っています。オフライン版ではプレイヤーの初期化はGameMainSceneが行っていましたが、CountDownSceneと機能を分担し、ソースコードの量を少なくしています。

SceneBace_MoveToメソッド内では、先述したGameDateTimeオブジェクトを利用して、今の時間とスタート時間の差をとってカウントの数字を計算しています。

ソースコード（CountDownSceneより抜粋）

```
Private count_ As Integer
  Public Sub SceneBase_MoveTo() Implements SceneBase.MoveTo
    If main.GameDateTime Is Nothing Then Return
    count_ = (main.StartTime.ToFileTime() - main.GameDateTime.Now.ToFileTime()) /
10000000
    'Debug.WriteLine("CountDownScene.PaintTo:Count=" & count_)
```

```
    If count_ <= 0 Then main.RaceStart()
```

その他についてはオフラインの初期化処理と同じなので解説を省きます。

　カウントダウンが終了するとRaceFormオブジェクトのRaceStartメソッドが呼び出されて、レースが始まるのでGameMainSceneへの切り替えを行います。なお、GameMainSceneオブジェクトについてはシーンの切り替えを瞬時に行えるように、あらかじめRaceForm_Loadイベントハンドラ内でゲーム起動時に初期化しています。

ソースコード（RaceFormより抜粋）

```
' レースゲームのスタート
  Public Sub RaceStart()
    Debug.WriteLine("RaceForm.GameStart(gamestartmessage)")

    ' タイマーの停止
    gameTimer_.Stop()

    ' カウントダウン画面をはずす
    If Me.Controls.Contains(countDownScene) = True Then
      ' Disposeする前に、DirectXPanelをRemoveして、いっしょにDisposesされないようにする。
      countDownScene.Controls.Remove(directXPanel_)
      Me.Controls.Remove(countDownScene)
      countDownScene.Dispose()
    End If

    ' レースゲームのシーンに切り替え。
    ' 切り替えをスムーズに行うためにGameMainSceneオブジェクトは事前に生成しておく。
    gameMainScene.Initialize(directXPanel_)
    gameMainScene.Visible = True

    scene_ = gameMainScene

    ' タイマーのスタート
```

```
    gameTimer_.Interval = 100
    gameTimer_.Start()
  End Sub
```

以上がカウントダウン処理になります。

・ 座標情報のやり取りとその他について

この段落ではまず、座標データの送受信についてのソースコードを記述しておきます。ここまでで解説したものなので、解説は省かせて頂きます。

ソースコード（GameMainSceneより抜粋）

```
' 座標データを全プレイヤーに送信
    Dim positionMessage As PositionMessage = New PositionMessage()
    positionMessage.Position = myPlayer_.Position
    main.SendMessage(positionMessage)
  End Sub
```

ソースコード（GamePeerより抜粋）

```
' メッセージを受信したときの処理
  Private Sub DataReceived(ByVal sender As Object, ByVal e As ReceiveEventArgs)
            ' 座標データを更新する
        main.UpdatePosition(e.Message.SenderID, CType(message, PositionMessage))
      Case MessageType.PlayerStatusMessage
```

ソースコード（RaceFormより抜粋）

```
' 座標情報の更新
  Public Sub UpdatePosition(ByVal playerID As Integer, ByVal positionMessage As
PositionMessage)
    SyncLock playerList_
      playerList_.GetPlayerByPlayerID(playerID).Position =
positionMessage.Position
    End SyncLock
  End Sub
```

全ソースコード（GameMainScene）

```
' レースゲームのシーン

Imports Microsoft.DirectX
Imports Microsoft.DirectX.Direct3D

Public Class GameMainScene
  Inherits System.Windows.Forms.UserControl
  Implements SceneBase
Private main As RaceForm
  Private directXPanel As DirectXPanel

  Public Sub New(ByRef main As RaceForm, ByRef directXPanel As directXPanel)
    Me.New()

    Me.main = main
    Me.directXPanel = directXPanel

    Me.Initialize(directXPanel)
  End Sub

  ' 初期化する
  Public Sub Initialize(ByRef directXPanel As directXPanel)
    Me.directXPanel = directXPanel

    'Controlの追加
    If Not Me.Controls.Contains(directXPanel) Then
      Me.Controls.Add(Me.directXPanel)
    End If

    time_ = 0
  End Sub

  Public ReadOnly Property PlayerList() As PlayerList
    Get
      Return Me.main.PlayerList
```

```
    End Get
  End Property

  Private myPlayer_ As Player
  Public Property MyPlayer() As Player
    Get
      Return myPlayer_
    End Get
    Set(ByVal value As Player)
      myPlayer_ = value
    End Set
  End Property

  ' ゲームループ
  Private Const TIME_LIMIT As Integer = 1000
  Private time_ As Integer = 0
  Private random_ As Random = New Random()
  Public Sub SceneBase_MoveTo() Implements SceneBase.MoveTo
    'Debug.WriteLine("GameMainScene.SceneBase_MoveTo:starttime " &
main.StartTime())
    'Debug.WriteLine("GameMainScene.SceneBase_MoveTo:now " & DateTime.Now())

    If main.StartTime > DateTime.Now Then Return

    time_ = time_ + 1
    myPlayer_.SpeedUp( -0.05)
    '制限時間を越えていたらゲームオーバー
    If time_ > TIME_LIMIT Then
      TimeUp()
    End If
    timeLabel.Text = String.Format("TIME {0}", time_)
    speedLabel.Text = String.Format("SPEED {0} km/h", CInt(myPlayer_.Speed * 100))

    'Playerの座標変換を行う
    Dim i As Integer
    For i = 0 To PlayerList.Count  - 1
```

```
      If PlayerList.GetPlayer(i) Is Nothing Then Exit For
      PlayerList.GetPlayer(i).MoveTo()
    Next i

    ' ゴール地点に到達したかどうかをチェック
    For i = 0 To PlayerList.Count - 1
      If PlayerList.GetPlayer(i) Is Nothing Then Exit For
      If PlayerList.GetPlayer(i).Position.Z < -52 Then
        main.GameOver()
      End If
    Next i

    ' 座標データを全プレイヤーに送信
    Dim positionMessage As PositionMessage = New PositionMessage()
    positionMessage.Position = myPlayer_.Position
    main.SendMessage(positionMessage)
  End Sub

  Public Sub SceneBase_PaintTo() Implements SceneBase.PaintTo
    '視点をユーザのキャラクターに合わせる
    directXPanel.GetDirect3DDevice.Transform.View =
Matrix.LookAtLH(myPlayer_.ViewPosition, myPlayer_.ViewTarget, New Vector3(0.0F,
1.0F, 0.0F))

    '再描画を通知
    directXPanel.Refresh()
  End Sub

  Private Sub TimeUp()
    ' GameOverSceneへ描画の切り替え
    main.GameOver()
  End Sub

  Private Sub ClickButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClickButton.Click
    'Debug.WriteLine("GameMainScene.ClickButton_Click")
```

```
    myPlayer_.SpeedUp(0.1)
  End Sub
End Class
```

以上になります。また、３DCGなどに関するクラスなど、この章で解説していない他のクラスは全て他の章で先述したものと同じなので、改めて解説は致しません。そのため、以上で研究の解説は終了となります。

終章

・ 感想と反省点について

私はこの研究について最初に一言で感想を述べると、「同じ方向でレベルアップしたものをつくりたい」ということです。やはり、３DCGとネットワークという一度に二つの大きなチャレンジということもあり、理解していくのに非常に時間がかかりました。しかし、製作を終えたというところで、やっとどういうものか掴んで来たので、むしろこれからオリジナリティあふれた作品に取り掛かれるようになった準備ができたという思いが非常に強いです。そして、反省点としては、とにかく応用してオリジナリティを付加できる時間も、キャパシティも、自分には一年では足りなかったということです。計画当初の狙いとしては、前進以外の動きや、シーンに応じたBGMや、他のプレイヤーとのチャット以外でのコミュニケーションなどができる様になることが希望でした。そして逆に、良かった点としては、とにかく基礎や基本だけだったとしてもネットワークと３Dに関わったプログラムが自分個人でもなんとか理解し、作成できたという事です。完全な完成とはいきませんでしたが、プログラミングの醍醐味に一年かけてやっと少し触れる事ができました。

そして最後に、一年間という短い間でしたが、親身になってのご指導のほど誠にありがとう
ございました。

２００５年２月１日

遠藤　正義

参考文献

『VisualBasic.net & DirectX9でネットワークゲームプログラミング』
著　安田　隆次　　DART社　　2003.8.30
『３DCGメタセコイア入門』
著　横枕　雄一郎　伊藤　真健　むつきはつか　共著
オーム社　　2004.7