GoDot によるプログラミング教育について

箕原辰夫[*1]
Email: *minohara@cuc.ac.jp*

*1: 千葉商科大学政策情報学部政策情報学科

◎Key Words　　GoDot, Python, GDScript, Unity, 3D 開発環境

1.　はじめに

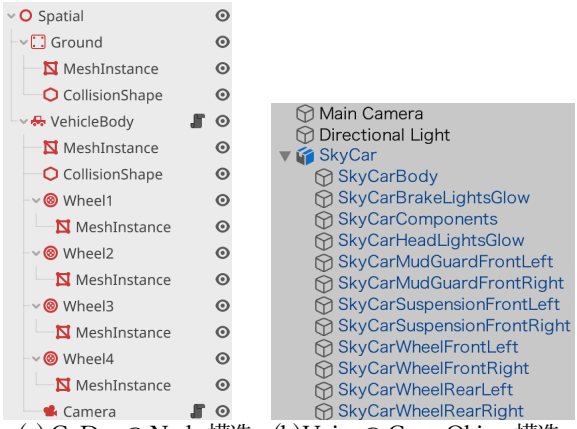
2022 年度秋学期において、ゼミナールの学生を対象に、GoDot[(1)]による 3D プログラミングの教育を行なった。この GoDot は、Unity[(2)]に似た 2D/3D ゲームや 3D アプリケーションのクロスプラットフォームでの開発環境になっているが、Unity と異なり、完全にライセンスフリーになっているのが大きな利点と考えられる。作成したプロジェクトをアプリケーションとして、iPhone なども含む様々なプラットフォームにリリースすることが可能で、そのときになんらのロイヤリティを課されることはない。この GoDot のプログラミングで利用できるのは、Python にかなり似た GDScript[(3)]と呼ばれるスクリプト言語、および、Unity と同様の C#が用意されているが、今回のゼミナールでは、プログラミングの授業で標準的に Python を教えているので、GDScript を選んだ。この報告では、GoDot と GDScript によるプログラミング教育の可能性について、実際に半期扱った経験から考察していきたい。Unity に慣れていると、最初は GoDot 特有の方式に戸惑うこともあったが、慣れると使い勝手も良かったので、今後も GoDot をゼミナールで利用していく予定にしている。

2.　GoDot と Unity との違い

GoDot は Unity と操作環境が非常に似ているが、Unity の感覚で使うと思いがけない落とし穴に陥る場合がある。ここでは、Unity との違いに焦点を当てて、開発環境の概略を紹介する。

2.1　コンポーネントがなくノードとして顕在化

Unity では、シーン上に配置されるのは、ゲームオブジェクト（GameObject）が基本であり、そこにコンポーネント（Component）を追加していく形になっている。コンポーネントの種類としては、良く使うものとしては、メッシュのレンダリングのためのコンポーネントや物理エンジン用のコンポーネント、あるいは C#などのスクリプトのコンポーネントなどがある。それに対して、GoDot では、Spatial（シーン空間を示すノード）以下のノード（Node）が基本のオブジェクトの単位になっており、その下に親子構造として、Unity ではコンポーネントに該当するようなノードを追加していく形になる。メッシュ構造(MeshInstance)や、衝突の物体構造（Collider）などがあるが、Unity と大きく異なるのは、逆にこれらを親の構造として、その下にメッシュ構造を子ノードとしてぶら下げる形になる。図 1 は自動車のノードを作成した例であるが、(a) にあるように、GoDot では衝突ノード (CollisionShape や WheelCollider) がノードとして現れており、その下に実際の 3D オブジェクトのメッシュがぶら下がる形になっている。ただし、スクリプトだけは、各ノードに添付できるようになっている。それに対して、Unity では、GameObject（これが 3D オブジェクトのメッシュ構造になっている）の階層構造だけが表れている。衝突のための情報は、コンポーネントとして GameObject に内包されているからである。



(a) GoDot の Node 構造　(b)Unity の GameObject 構造

図 1 Node と GameObject

このような構造は、Blender[(4)]や Autodesk Maya[(5)]などの 3D モデリングソフトウェアで作成した 3D オブジェクトデータをインポートするときに逆に不利になると思われる。Unity のような形にしておくと、それらの 3D モデリングソフトウェアで作成したオブジェクト間の階層構造をそのまま取りこめる利点がある。

2.2　標準 MeshInstance の作成

Unity の場合は、新規 3D オブジェクトのプリミティブを追加する場合に、GameObject メニュー>>> 3D Object サブメニューで、立方体（Cube）や円筒（Cylinder）、球（Sphere）などをシーンの中に簡単に追加できるが、GoDot で同じことをする場合は、かなり面倒になる。まず、左側のシーンの階層構造のパネル（図 1 の（a）のパ

ネル）で、親のノードを選んで、ポップアップメニューの「子ノードを追加」を選択する。子ノードの種類として現れるパネルにおいて、図2のように標準の MeshInstance は、遥か下方の構造に隠れているので「Mesh」で検索して出す必要がある。

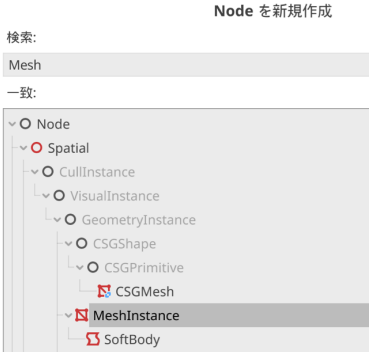


図2 遥か下方に隠れている MeshInstance

この MeshInstance をシーンに追加すると、Unity と同様にこのノードに関してのプロパティが右側のパネルに表示される。このプロパティの「Mesh」の部分が「空」になっているので、このメニューをクリックして、図3のように標準の 3D の Mesh の形状を選択する。

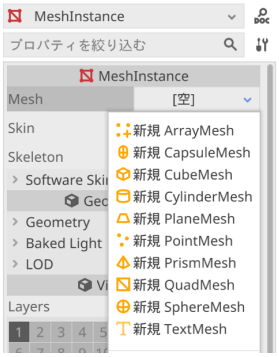


図3 標準の MeshInstance の形状選択

2.3 インポートできる 3D モデルの形式

GoDot でインポートできる 3D モデルの形式は、表1に表わされているファイル形式に限られており、これは Unity に比べると少し見劣りがする。

表1 インポート可能な 3D モデルの形式

ファイル形式	備考
glTF 2.0	テキスト形式・バイナリ形式の両方をサポート（Blender プラグインでエキスポート可能）
DAE(COLLADA)	かなり古い形式
OBJ (Wavefront)	Autodesk Maya のファイル形式
ESCN	GoDot のファイル形式（Blender からプラグインでエクスポート可能）
FBX	Open Asset Import library が必要

Unity では Blender のファイル形式（.blend 拡張子で終わる）を直接インポート可能であり、Blender 上で作成したアニメーションなども含めてインポート可能なのに比べると、だいぶ見劣りがする。glTF や ESCN 形式は、Blender 側でプラグインをインストールしておく必要があり、特に ESCN 形式のプラグインは、GitHub から下記のアドオンをダウンロードして Blender 側にインストールする必要がある。

https://github.com/godotengine/godot-blender-exporter

Unity のプレファブ(Prefab)を利用したアセット(Asset)は、非常に多く作成されて公開されており、アセットで市場を形成しているぐらいになっている。それに対して、GoDot のアセットは、ほとんどが無料のアセットで、質も量も Unity のアセット市場と比べて、かなり見劣りする内容になっている。

3. GDScript の記述方法

Unreal[6]は、データフロー的に計算ノードを結合してスクリプトを記述するが、プログラミング教育としてはあまり効果がないように思える。もちろん、Unreal では C++でのスクリプト記述も可能であるが、リファレンスは Unity や GoDot ほどは充実していない。豊富なリファレンスが存在しており、通常のプログラミング言語と同様な記述方法で、各オブジェクトの振る舞いを記述できる方が、プログラミング教育の観点からは教育効果が高いように感じられる。Unity は、そのような意味で C#の文法規則に従ってスクリプトを記述し、それをスクリプト読込み時にコンパイルしてエラーなどを表示するので通常のプログラミング教育に直結するように感じられる。GoDot も同様であり、C#も利用可能であるが、通常のプログラミング言語として Python を標準で教えている状況から、Python に非常に近い文法構造を持つ GDScript を利用することにした。

Unity では、プログラミングのエディタ環境は、Visual Studio や VSCode などの外部エディタをインストールして選ぶ形になるが、GoDot では、組込みでエディタが内包されているので、これは Unity に比べてコンパクトで扱いやすいのではないかと思われる。

3.1 GDScript の文法概略

GDScript の文法[3,7]は、ほぼ Python と同様なため、Python と異なる点だけを説明していく。

- 変数の導入には、var 宣言が必要
  例：**var** x = 12
- 定数型の変数を宣言するための const がある
  例：**const** E = 2.718281828

・ Python では 3.10 から導入された match 文を標準で持つが case 句はなく、パターンを直接記述する
例：**match** typeof(x):
TYPE_STRING: print("string")
・ 関数宣言は、def ではなくて、func を用いる
例：**func** some_function(param1, param2): **pass**
・ コンストラクタは、__init__ではなくて、_init
・ サブクラスから、スーパークラスの同名の関数（メソッド）を呼び出すためには、super を使うのではなくて、ドットを利用する
例：.overriden_function(param)
・ C++や C#と同様に列挙型は標準装備されている
例：**enum** State {IDLE, RUN=1, WAITING}
・ クラスを拡張するには、extends 文を用いる
例： **extends** Animal

これ以外にも細かな違いはあるが、Python でプログラミングできる学生であれば、問題なく、移行できるレベルである。Unity は、かなり昔の版では、C#と JavaScript の他に、Boo[8]という Python に酷似していた静的型付け言語でプログラミングを行なうことができた。程なくして、Unity の開発言語としては Boo のサポートが中止になった経緯がある。初期の Unity を教えていたときは、Boo を利用したこともある。Boo ほどではないにせよ、Python 流のプログラミング言語を用いて 3D 開発環境でプログラミングできるのは、Panda3D[9]のライブラリ以来だと思われる。

3.2 オブジェクト操作関数と Transform・物理操作

Unity では、Controller の C#のスクリプトがクラスと共に生成され、その中に Start メソッドと Update メソッドが標準で用意されており、最初に Start メソッドが呼び出され、各フレームの描画毎に Update メソッドが呼び出される仕組みになっている。GoDot でも、同じような仕組みになっているが、まず、スクリプトではクラスの定義の代わりに、Node クラスの拡張を宣言する必要がある。Start メソッドの代わりに_ready 関数が、Update メソッドの代わりに_process 関数が用意されている。アンダーバー（_）で名前が始まるのは、Python 流に解釈すると外部非公開関数であると考えられる。Unity では、Time.deltaTime で経過時間を参照する必要があったが、_process 関数では、delta 引数で始めから経過時間が与えられている。

```
extends Node
# ノードがシーンに加わるときに最初に呼ばれる
func _ready(): pass
# 毎フレームごとに呼ばれる
# 'delta' は、前回のフレームからの経過時間
func _process(delta): pass
```

シーンの座標系は、Unity と同じで y 軸が頭頂方向で、z 軸が奥行き方向になっている。オブジェクトを移動させる Transform クラスは、ほぼ Unity と同様のものが用意されている。以下は、カメラノードの制御スクリプトの中でターゲットとなるオブジェクトの方向を見るような関数の定義になっている。

```
func setPosition( target. radius, phi, theta ):
  targetpos = target .get_translation( )
  var x = radius * cos( phi ) * cos( theta )
  var y = radius * sin( phi )
  var z = radius * cos( phi ) * sin( theta )
  self.set_translation(Vector3( x, y, z ) + targetpos )
  self.look_at( targetpos, Vector3( 0, 1, 0 ) )
```

オブジェクトの物理的な挙動を扱うためには、Unity のように RigidBody のコンポーネントの中で値を設定することはできず、RigidBody ノードのコントロールスクリプトの中で、重力方向を指定する。_integrate_forces 関数の中で指定するが、この関数に与えられる引数の state は、経過時間ではなく、剛体に働く物理環境の状況を表わす内容を保持している。

```
extends RigidBody
func _integrate_forces( state ):
    state.add_central_force( Vector3( 0.0, -9.8, 0.0 ) )
```

物理的な挙動を正確に扱うために、フレームレートに依存しないフレーム間の経過時間が必要な場合は、_process 関数ではなく、_physics_process 関数を使う。この中で、物理的な衝突や跳ね返りを扱う。

```
extends KinematicBody
var velocity = Vector3(250, 0, 250)
func _physics_process( delta ):
    var collision_info =
        move_and_collide( velocity * delta )
    if collision_info:
        velocity =
         velocity.bounce( collision_info.normal )
```

3.3 WheelCollider のプログラミング

自動車の 3D オブジェクトを平面上に置き、各タイヤに WheelCollider をつけて（GoDot の場合は Unity とは逆で、WheelCollider の下に 3D メッシュのノードを作成する）、駆動させる方式は、Unity と GoDot ではほぼ同じ仕組みなので、同じ考え方で記述できる。以下は、GoDot で、記述した例であるが、Unity の WheelCollider についている例題の簡易版になっている。なお、該当関数は通常の_process 関数ではなく、物理関係を扱うために、_physics_process 関数を利用する。engine_force の値と

steering の値を変えれば、車が動く。

```
extends VehicleBody
var accelSpeed = 5
var rotateSpeed = deg2rad(15)
var maxRotate = deg2rad(30)
var maxAccel = 20

# 毎フレームごとに呼ばれる
func _physics_process(delta):
    if Input.is_action_pressed("ui_up"):
        engine_force += accelSpeed * delta
        if engine_force > maxAccel:
            engine_force = maxAccel
    if Input.is_action_pressed("ui_down"):
        engine_force -= accelSpeed * delta
        if engine_force < -maxAccel:
            engine_force = -maxAccel
    if Input.is_action_pressed("ui_left"):
        steering += rotateSpeed * delta
        if steering > maxRotate:
            steering = maxRotate
    if Input.is_action_pressed("ui_right"):
        steering -= rotateSpeed * delta
        if steering<-maxRotate
            steering=-maxRotate
```

3.4 ゲームサーバのプログラミング

GoDot では、Unity と同様にサーバを置いて、RPC（Remote Procedure Call）で通信をしながら、クライアント・サーバ間で複数のプレイヤーを対象にしたオンラインゲームを作成する環境が用意されている[10]。まず、サーバー側は接続プレイヤー数の上限を制限して、ポートを指定したサーバをピア（peer：ネットワーク上のプレイヤーを示すオブジェクト）上に用意する。

```
var peer = NetworkedMultiplayerENet.new()
peer.create_server(SERVER_PORT,
MAX_PLAYERS)
get_tree().network_peer = peer
```

クライアント側は、ピア上にサーバの IP アドレスとポート番号を指定してクライアントを用意する。

```
var peer = NetworkedMultiplayerENet.new()
peer.create_client(SERVER_IP, SERVER_PORT)
get_tree().network_peer = peer
```

サーバ側の関数は、rpc 関数によってクライアント側から呼び出すことが可能になっている。また、rpc_id で特定の相手の関数を呼び出すことも可能である。また、RPC で呼び出される関数の定義には、remote タグを追加しておく必要がある。

```
rpc("関数名", <関数へのオプショナル実引数>)
rpc_id(ピア ID,"関数名", <オプショナル実引数>)
remote func 関数名( 仮引数 ): pass
```

remote 以外に、ホストのピアだけ実行される master、および、ホスト以外のピアだけ実行される puppet のタグも追加することが可能になっている。

4. おわりに

ゼミナールでは、Unity を長年利用してきたが、プログラミング言語教育として科目として教えている Python との親和性、および完全ライセンスフリーなどの状況を考慮して、2022 年度秋学期から GoDot も利用してみることにした。Unity を扱ってきた教員であれば、移行は割とスムーズに行なえるのではないかという印象を持った。ただし、アセットの豊富さなどは Unity より見劣りする部分があるので、そのような外部アセットを利用するようなプロジェクトでは、Unity を使い続けるしかないだろう。Unity も引き続きゼミナールでは利用していく予定であるが、GoDot はプログラミング教育環境としては捨てがたいものになってきており、これからも GoDot も並行して扱う予定にしている。

参考文献

(1) “GoDot,” GoDot community, https://godotengine.org, (2007-2023).
(2) “Unity,” Unity Technologies, https://unity.com/ja, (-2023).
(3) “GDScript,” Andrew Wilke et al., https://gdscript.com, (-2023).
(4) “Blender,” Blender Foundation, https://www.blender.org, (-2023).
(5) “Autodesk Maya,” Autodesk Inc., https://www.autodesk.co.jp/products/maya/, (-2023).
(6) “Unreal Engine 5,” Epic Games, https://www.unrealengine.com/ja/unreal-engine-5, (-2023).
(7) “GDScript の基本,” GoDot Community, https://docs.godotengine.org/ja/stable/tutorials/scripting/gdscript/gdscript_basics.html, (-2023).
(8) “Boo: A scarily powerful language for .Net,” Rodrigo B. de Oliveira, https://boo-language.github.io, (2009-2018).
(9) “Panda 3D,” Carnegie Mellon University, https://www.panda3d.org, (2019-2023).
(10) “High-level multiplayer,” GoDot Community, https://docs.godotengine.org/ja/stable/tutorials/networking/high_level_multiplayer.html, (-2023).