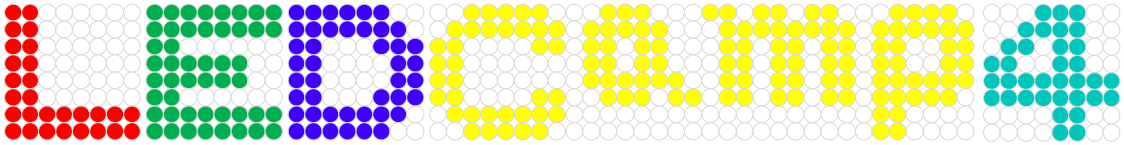
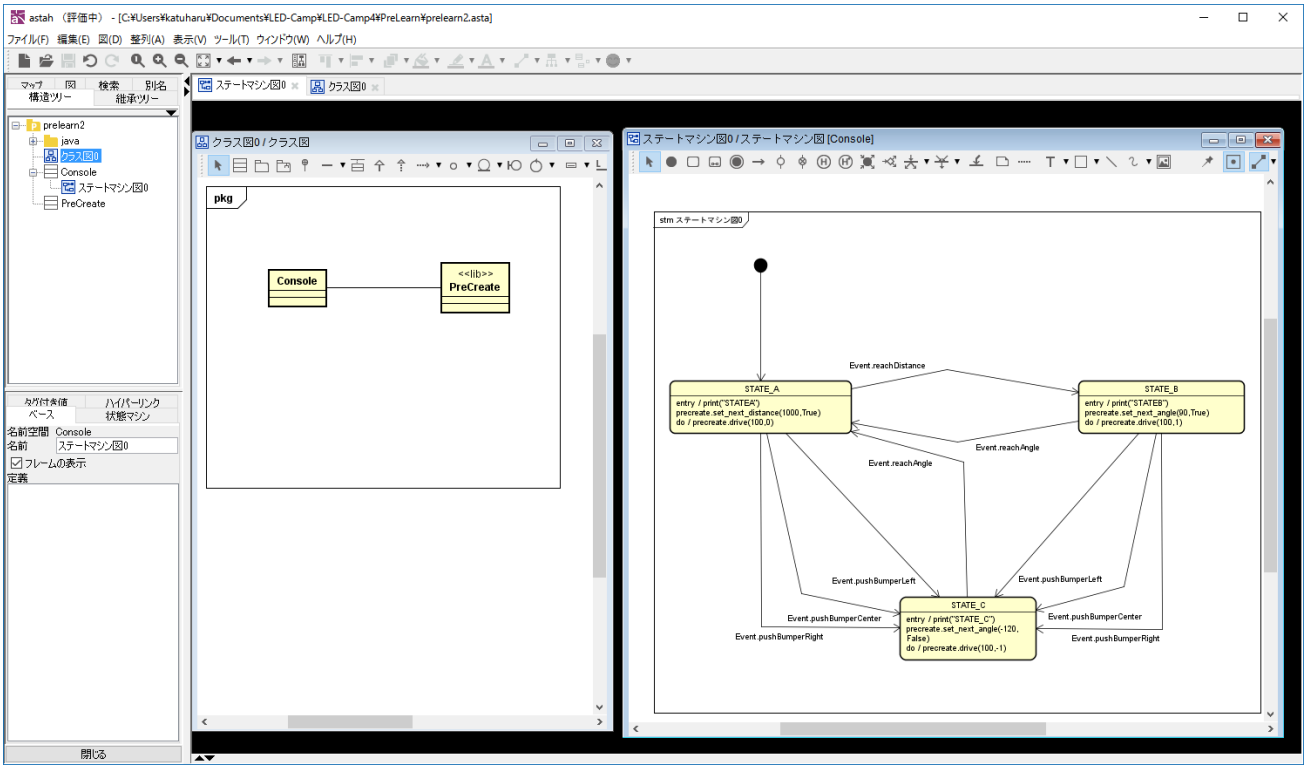


MDD チュートリアル

岡山 直樹

1 内容

1 MDD チュートリアル概要



当日のモデル駆動開発（MDD）演習では、astah* professional 上で UML のクラス図やステートマシン図を作成して開発を進めていきます。

そこで事前実習として MDD チュートリアルでは、astah* professional や m2t プラグインの簡単な使い方を覚えていただき、MDD が行えるようになるために小規模な開発を行い実行してもらいます。

m2t プラグインは、クラス図、ステートマシン図の 2 つのモデル図からソースコードを生成します。

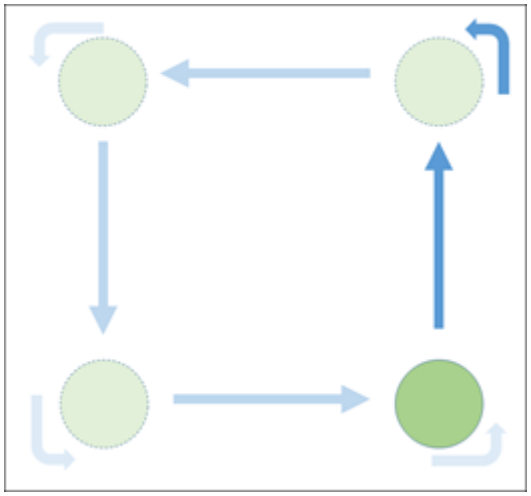
Naoki Okayama

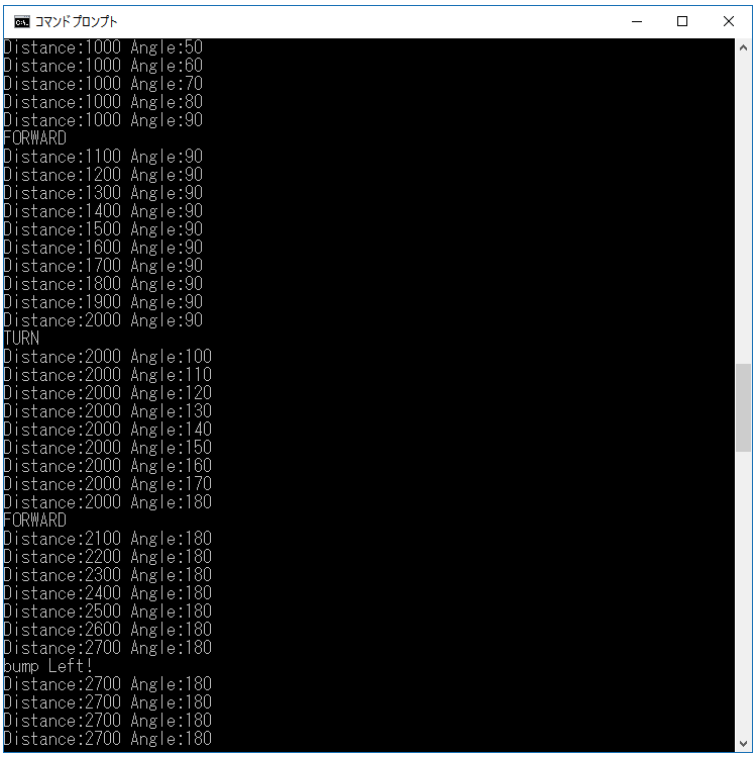
2 PreCreate2

当日の実習では、掃除機型ロボット Create2 を使用します。今回の事前実習では、まだ皆さんのお手元には Create2 がありません。そこで、PC 上で簡易的に動く、PreCreate2 を用いて事前実習を行います。PreCreate2 は、コンソール上に、現在の進んだ距離、現在角度が見られるだけの簡素なものとなり、動く動作も、単純な前進・回転のみとなります。また、PreCreate2 はたまに障害物にぶつかり、停止してしまいます。

この PreCreate2 を用いて、部屋の角をぐるぐる回るようなソフトウェアを事前実習では開発します。

「前進→90 度回転→前進→90 度回転・・・・」のようなイメージです。

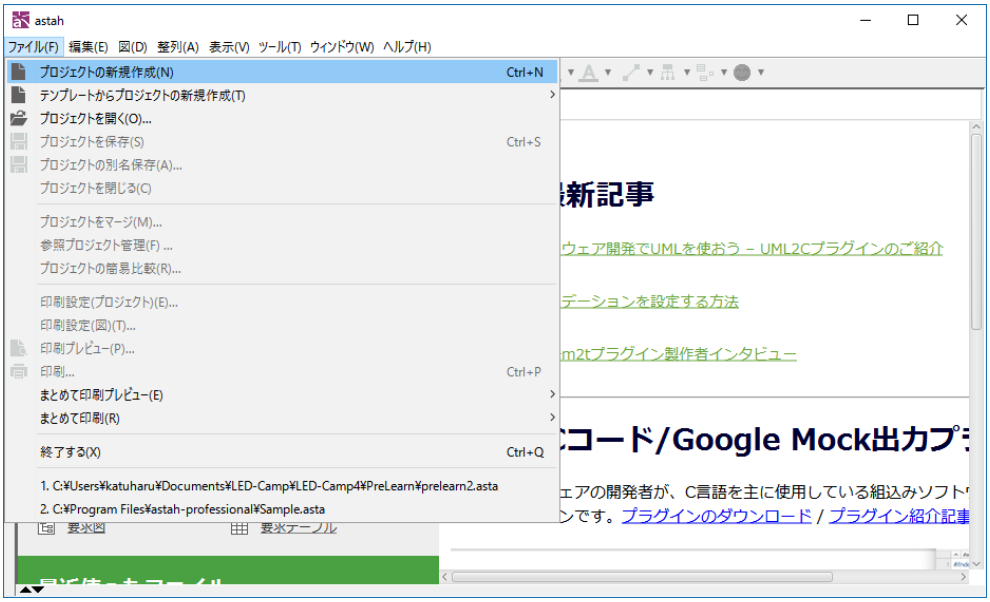




Naoki Okayama

3 プロジェクトを作る

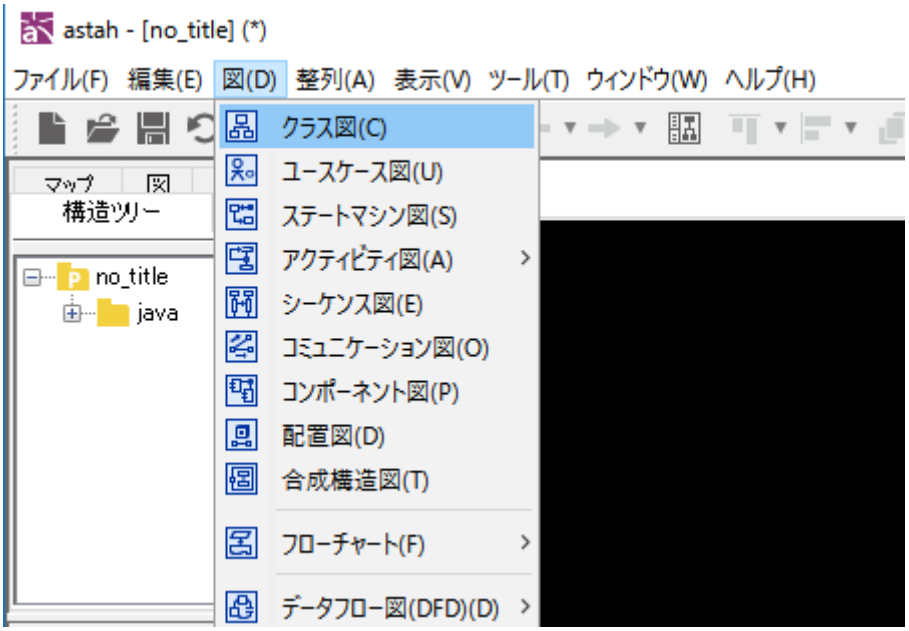
まず、プロジェクトファイルを作成します。astah を起動して、「ファイル → プロジェクトの新規作成」を選択し、プロジェクトファイルを作成します。プロジェクトの保存は「ファイル → プロジェクトの保存」を選択します。任意のフォルダに保存をしてください。



4 クラス図を作ってみよう

4.1. クラス図の追加

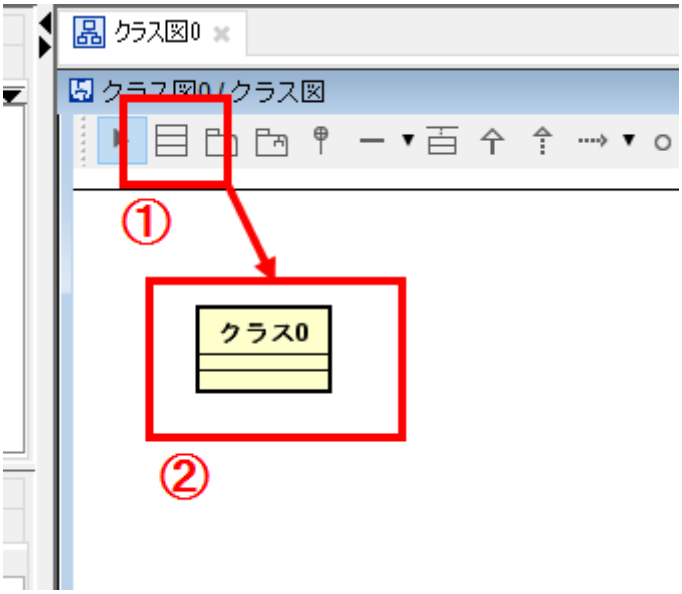
「図 → クラス図」を選択し、新規クラス図を作成します。

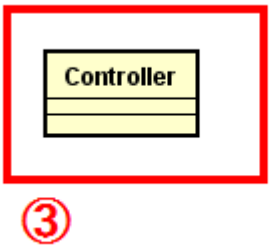


Naoki Okayama

4.2. クラスの追加

新規クラス図を作成したら、クラスを追加してクラス図を作成していきます。





① まず、クラスを選択します。

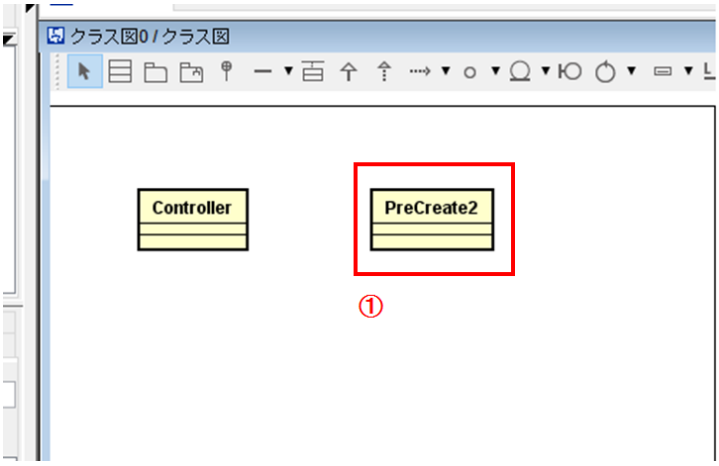
② キャンバスをクリックしてクラスを配置します。

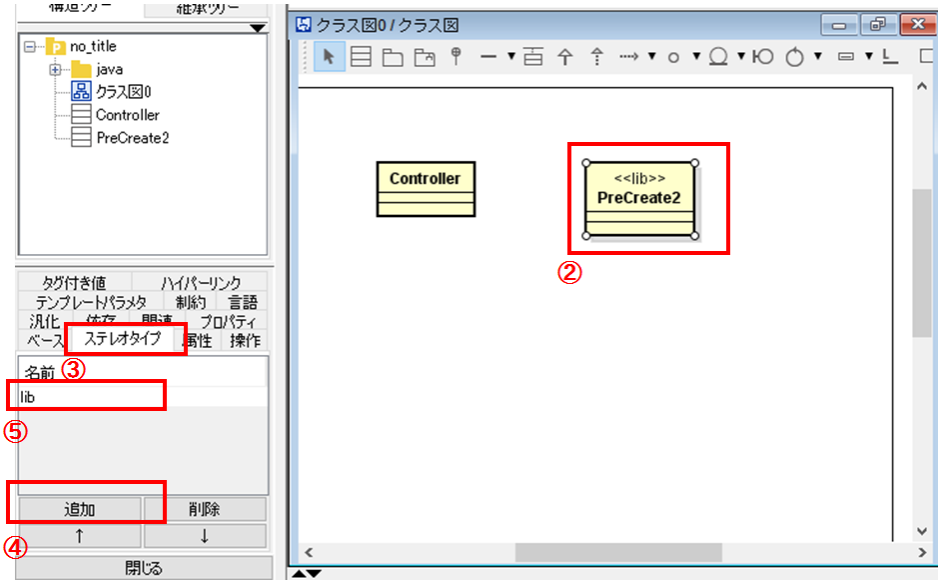
③ 「クラス 0」と書かれている部分をクリックするとクラス名を自由に変更できます。ここでは「Controller」とします。

4.3. ライブラリの追加

次に、ライブラリを追加します。今回の事前実習で記述するのは Controller のモデルで、実際の動作は実行委員の用意した PreCreate2 が行います。PreCreate2 はすでにコードが存在するため、コード生成を行う必要はありません。そこで、ライブラリであることをステレオタイプで指定する必要があります。

7





① 先ほど「Controller」クラスを追加した手順と同じように、「PreCreate2」クラスをクラス図に配置します。

② 次にステレオタイプを追加します。先ほど作成した「PreCreate2」クラスを選択します。

Naoki Okayama

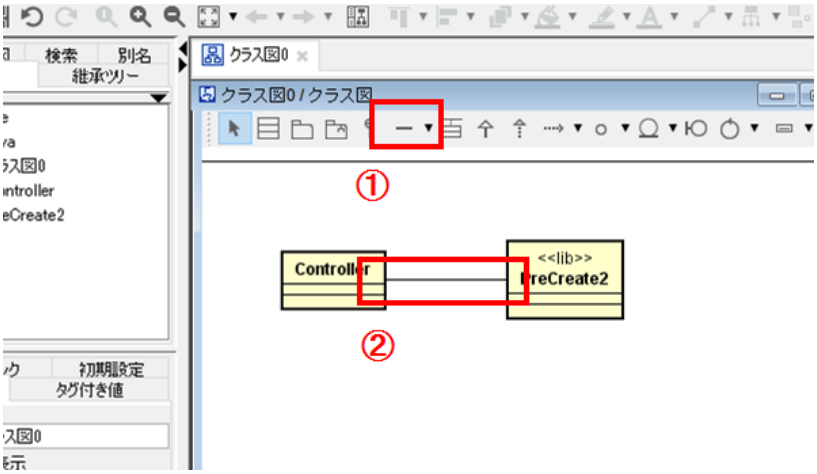
③　左側のビューのステレオタイプを選択します。

④　追加をクリックします。

⑤　「lib」と入力します。

図のように、「PreCreate2」クラスの名前の上に「<<lib>>」と表示がされればステレオタイプの設定は終わりです。

4.4.　関連の追加



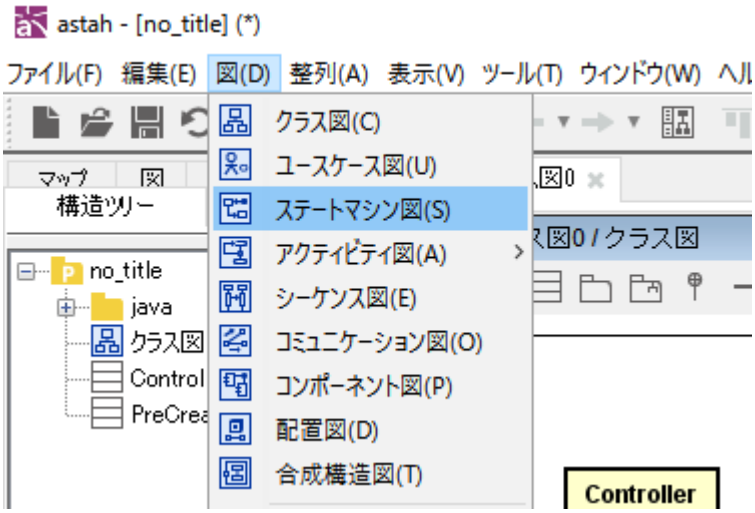
①　関連を選択します。

②　「Controller」と「PreCreate2」を接続します。

5 ステートマシン図を作ってみよう

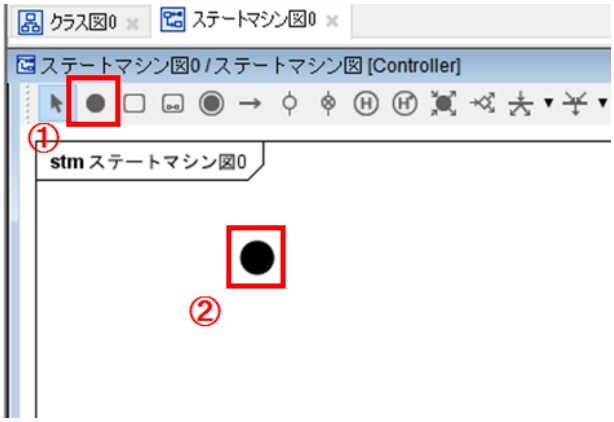
5.1. ステートマシン図の追加

「図 → ステートマシン図」を選択し、新規ステートマシン図を作成します。



5.2. 開始疑似状態の追加

まずは、ステートマシン図の起点を表す、開始疑似状態を追加します。



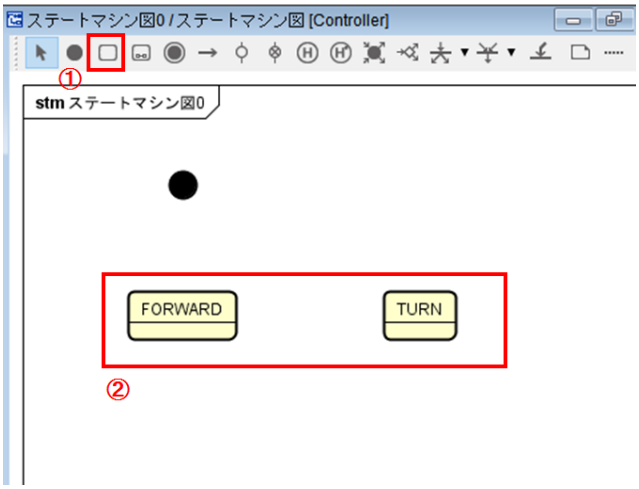
① 開始疑似状態を選択します。

② ステートマシン図上に、開始疑似状態を配置します。

Naoki Okayama

5.3. 状態の追加

開始疑似状態を追加したら、状態を追加していきます。



① 状態を選択します。

② ステートマシン図上に、状態を配置します。

今回は、前進をする状態の「FORWARD」と、回転をする状態の「TURN」の 2 つを追加してください。状態名の変更はクラス名を変更したときと同様に、状態名をクリックすると自由に変更ができます。

5.4. 状態のアクティビティの指定

状態の内部には、アクティビティと呼ばれる処理を指定することができます。このアクティビティの設定を行い、その状態内でどのような動作を行うか決めていきます。
状態のアクティビティには、「入場動作（entry）」「実行活動（do）」「退場動作（exit）」の 3 つが指定できます。

- 入場動作（entry）

  状態へ遷移されたタイミングで実行されるアクティビティです。

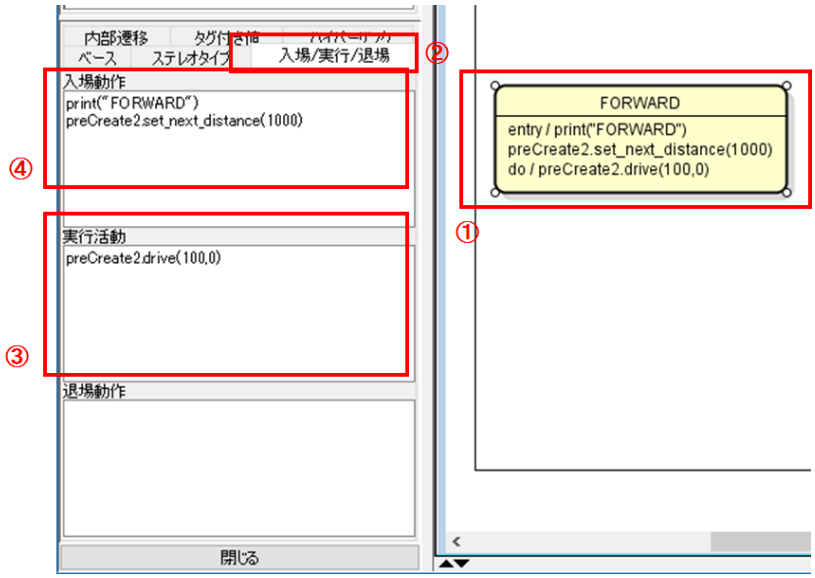
- 実行活動（do）

  状態内で実行されるアクティビティです。

- 退場動作（exit）

  その状態からでるタイミングで実行されるアクティビティです。

これらのアクティビティを指定していきます。
今回の実習では、モデル図からコード生成する際、Python 言語のコードを出力します。そのため、アクティビティは Python 言語で指定します。



① アクティビティを指定する状態を選択します。

② 左側のビューの「入場/実行/退場」タブを選択します。

③ まず、実行活動を指定します。「FORWARD」状態では、PreCreate2 を前進させます。

PreCreate2 の API として、「drive」関数を用意してありますので、これを使用します。

【関数名】	drive(velocity, radius)
【動作・意味】	PreCreate2 を指定の移動速度および回転方向で動作させる。
【引数】	velocity：移動速度（radius=0：1 周期の移動距離　radius!=0：1 周期の回転角度） radius：回転方向（0：直進　1：+方向　2：-方向）

【返値】	無し

m2t プラグインではクラス図に描かれたクラス名の先頭を小文字にしたインスタンスを自動的に生成します。そのためここでは、「preCreate2.drive(100,0)」と設定します。

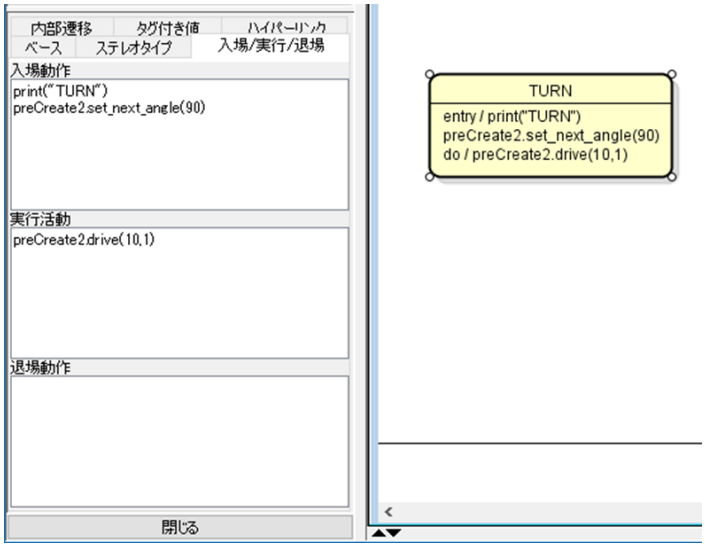
④ 次に入場動作を指定します。入場状態では、現在の状態の出力と、走行距離を設定します。現在の状態の出力は python の print 関数を使用します。

「print(“FORWARD”)」としてください。

走行距離を設定は、PreCreate2 の API の「setNextDistance」関数を使用します。この API を使用して走行距離を設定すると、設定した距離を走行すると「Event.reachDistance」というイベントを発生させます。このイベントをトリガとして今後、状態の遷移を行っていきます。

【関数名】	setNextDistance(distance)
【動作・意味】	指定した距離を移動した際にイベント Event.reachDistance を発生させる。
【引数】	distance：距離の指定
【返値】	無し

ここでは 1000 進むと次の状態へ遷移させるようにしたいと思います。そのため、「preCreate2.setNextDistance(1000)」としてください。



「FORWARD」状態のアクティビティを指定したら、「TURN」状態のアクティビティを指定します。「TURN」状態では、+方向に 10 度ずつ回転することにします。また、90 度回転したら、次の状態へ遷移します。「setNextDistance」関数と同様に、回転角度の設定は、PreCreate2 の API の「setNextAngle」関数を使用します。

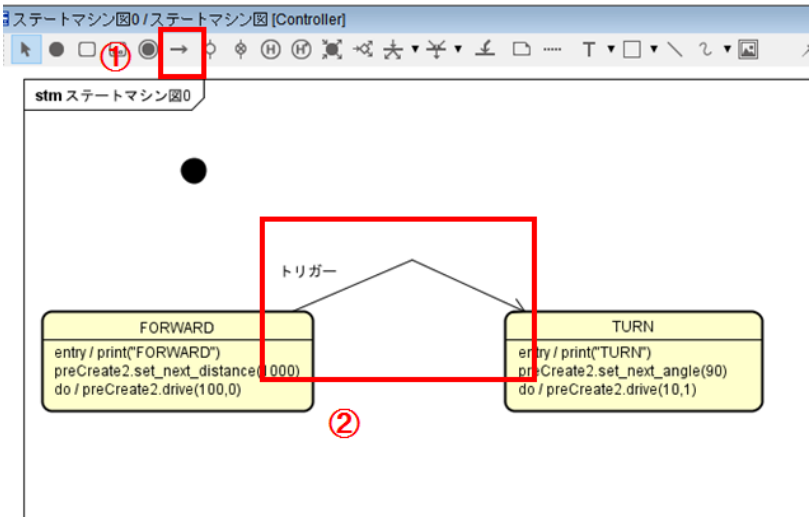
【関数名】	setNextAngle(angle)
【動作・意味】	指定した距離移動した際にイベント Event.reachAngle を発生させる。
【引数】	angle：回転角度の指定
【返値】	無し

これまでの、情報を元に以下のように指定してみてください。

入場動作：	print("TURN") preCreate2.set_next_angle(90)
実行活動：	preCreate2.drive(100,1)
退場動作：	なし

5.5. 遷移の追加

状態を配置し終えたら、遷移を追加します。



① 遷移を選択します。

② 「FORWARD」と「TURN」を接続します。接続した遷移は、中ほどをドラッグすると折り曲げることができます。

5.6. 遷移のアクティビティの指定

遷移を追加したら、遷移のアクティビティを指定します。

遷移のアクティビティには「トリガー」、「ガード」、「アクション」の 3 つがあります。

- トリガー

  状態が遷移するきっかけとなるイベントを指定します。

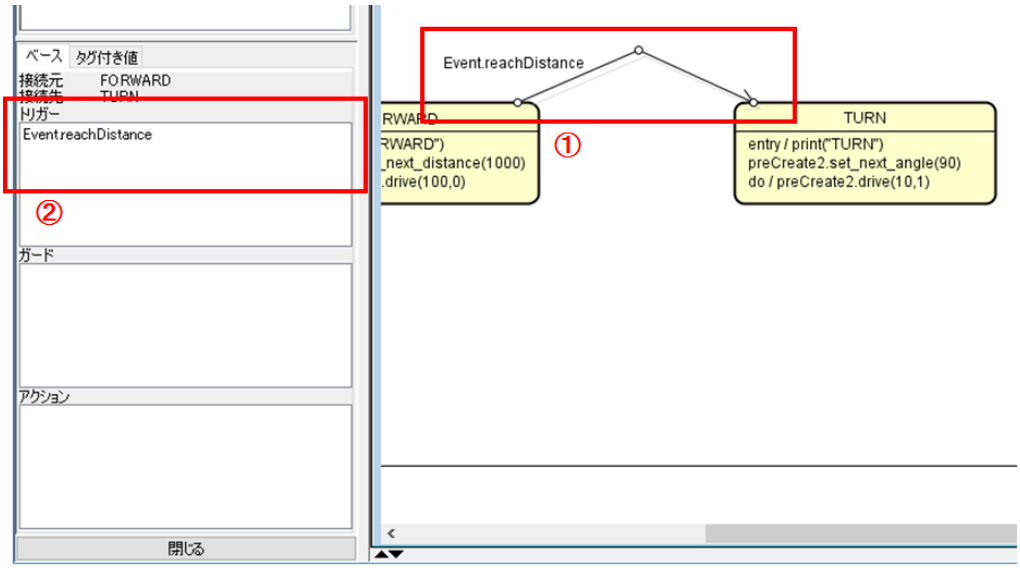
- ガード

  トリガーに指定したイベントが発生したときに、ガードに指定した条件が真であった場合は遷移をします。偽であった場合は、遷移をしません。

- アクション

  アクションは、遷移が起きるときに実行されます。

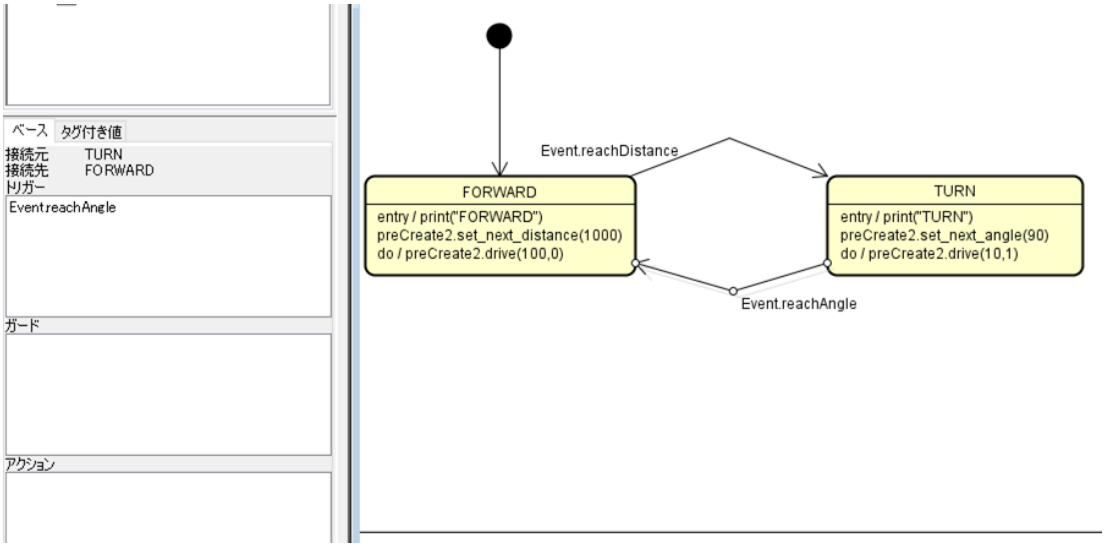
これらのアクティビティを指定していきます。



① アクティビティを指定したい遷移を選択します。

② 今回は、トリガーに「Event.reachDistance」を指定します。「FORWARD」の入場動作で「setNextDistance」で走行距離を設定しました。走行距離分走行すると「Event.reachDistance イベント」が発生するので、トリガに「Event.reachDistance」を指定すると遷移することができます。

これで、「FORWARD」から「TURN」への遷移ができました。

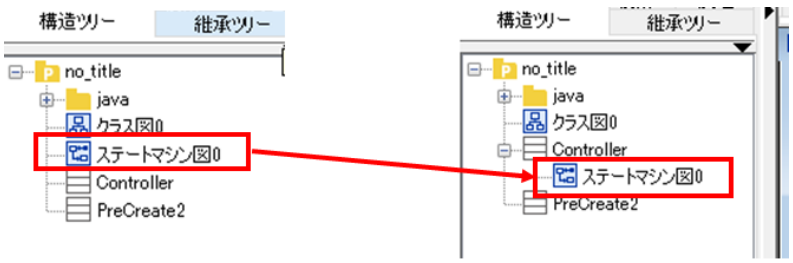


上記の図のように、「TURN」から「FORWARD」までの遷移の追加も行ってください。「TURN」から「FORWARD」へのトリガは「Event.reachAngle」となります。
また開始疑似状態から「FORWARD」への遷移も追加してください。こちらにはアクティビティは必要ありません。

6 コード生成をしよう

これでモデルは完成しました。これから、このモデルから、ソースコードを生成します。

6.1. クラスと状態の紐づけ



今回使用する m2t プラグインでは 1 クラスあたり、1 ステートマシンを持つようなコードを生成します。そこで、いま作成したステートマシン図と、クラス図を紐づけます。左側のビューに追加されたステートマシン図を「Controller」クラスの直下に配置されるようにドラッグアンドドロップします。これにより、「Controller」クラス内に、先ほど作成したステートマシン図のコードが生成されるようになります。

これを行わないと、正常にコードを生成することができませんので、忘れずに行うようにしてください。

6.2. テンプレートの追加

https://swest.toppers.jp/LED-Report/Camp4/files/MDD_PreLearn.zip

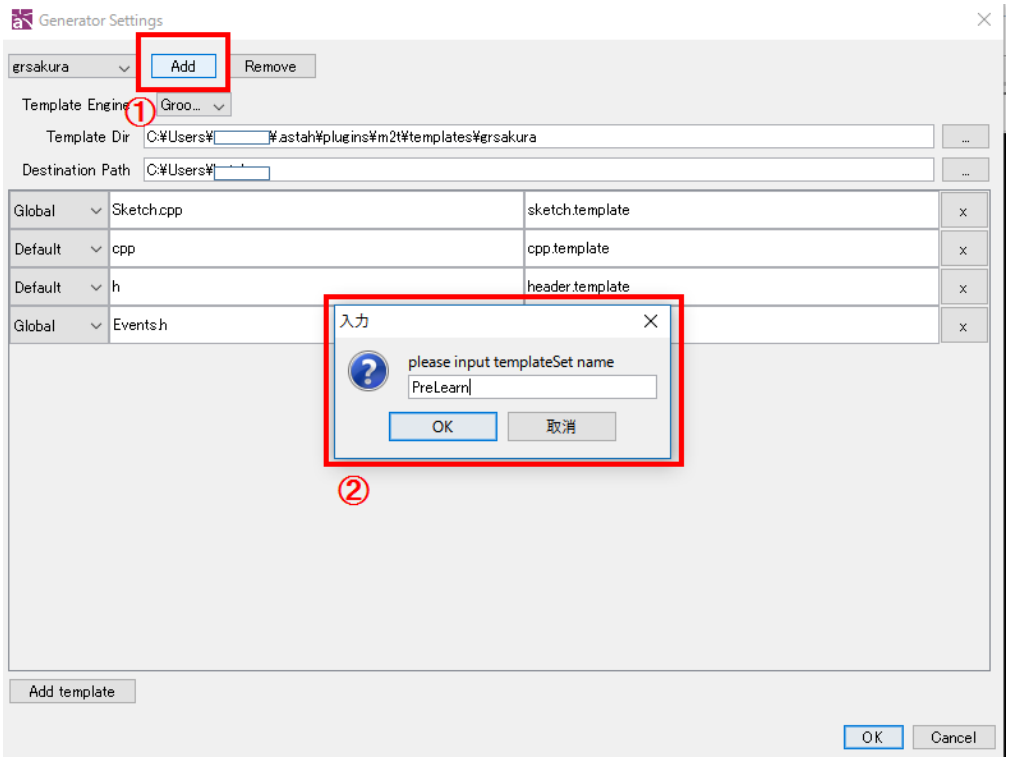
上記の URL へアクセスして、テンプレートファイルと、ライブラリをダウンロードしてください。

展開すると「template」フォルダの下に「PreLearn」というフォルダがあります。これがテンプレートファイルとなります。

「C:¥Users¥○○○¥.astah¥plugins¥m2t¥templates」へ、「PreLearn」フォルダをフォルダごとコピーしてください。

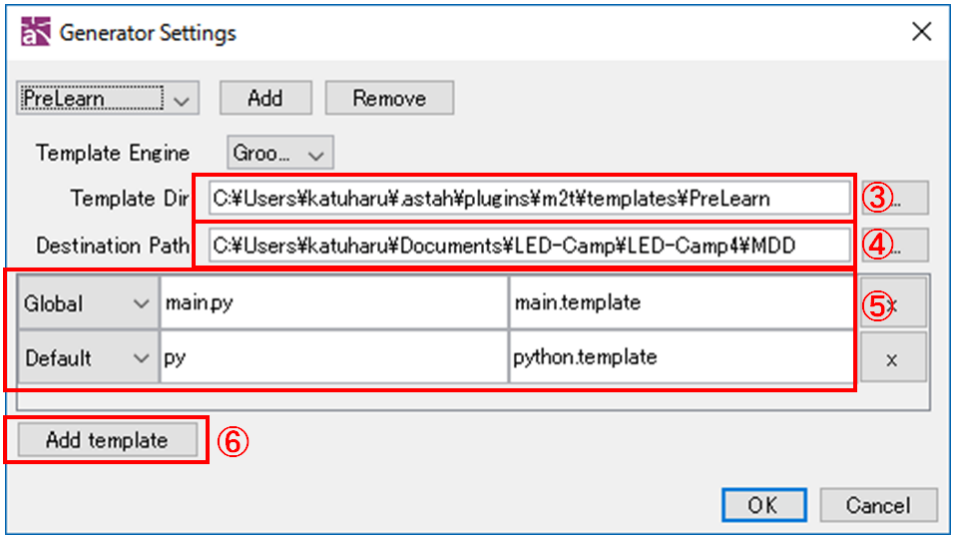
6.3. M2T プラグインの設定

コード生成を行う m2t プラグインの設定を行います。「ツール」→「m2t」→「setting」を選択して設定画面を開いてください。



① Add をクリックして新しい設定を作成します。

② ダイアログが出るので、「PreLearn」と入力してください。



③ テンプレートファイルの場所を指定します。先ほどテンプレートファイルをコピーしたディレクトリのパスを指定してください。

④ コードを生成するディレクトリを指定します。ダウンロードしてきたライブラリのディレクトリを指定してください。

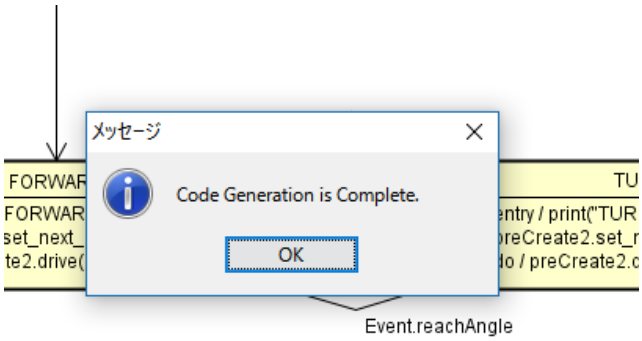
⑤ 「Add template」をクリックすると、拡張子や、ファイルごとにどのテンプレートファイルでコード生成を行うかの設定を追加できます。ここでは 2 つ設定を作成します。

⑥ 設定を入力します。今回は、実行をつかさどる「main.py」のテンプレートファイルと、実際にモデル図上に配置したクラスの生成「py」のテンプレートファイルがあります。それぞれ図のように設定を行ってください。一番右の欄はクリックするとファイルを選択するダイアログが表示されるので、先ほどコピーしたテンプレートファイルをそれぞれ指定してください。

6.4. コード生成

設定が終わったので、コード生成を行います。「ツール」→「m2t」→「Generate」を選択します。

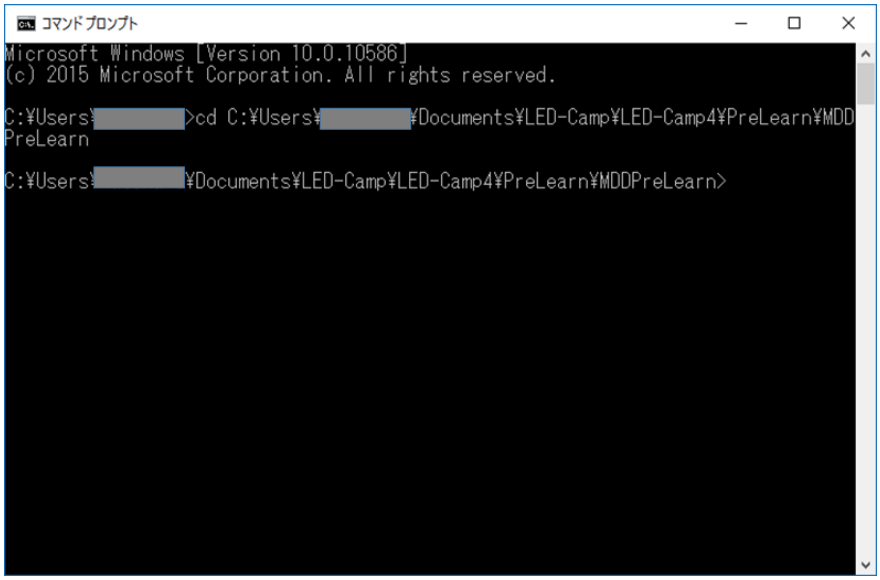
下図のように表示されれば生成成功です。もし下図のように表示されなかった場合は、モデル図の接続や、プラグインの設定、6.1 章のクラスと状態の紐づけを見直すか、astah を再起動して再度生成を試みてください。



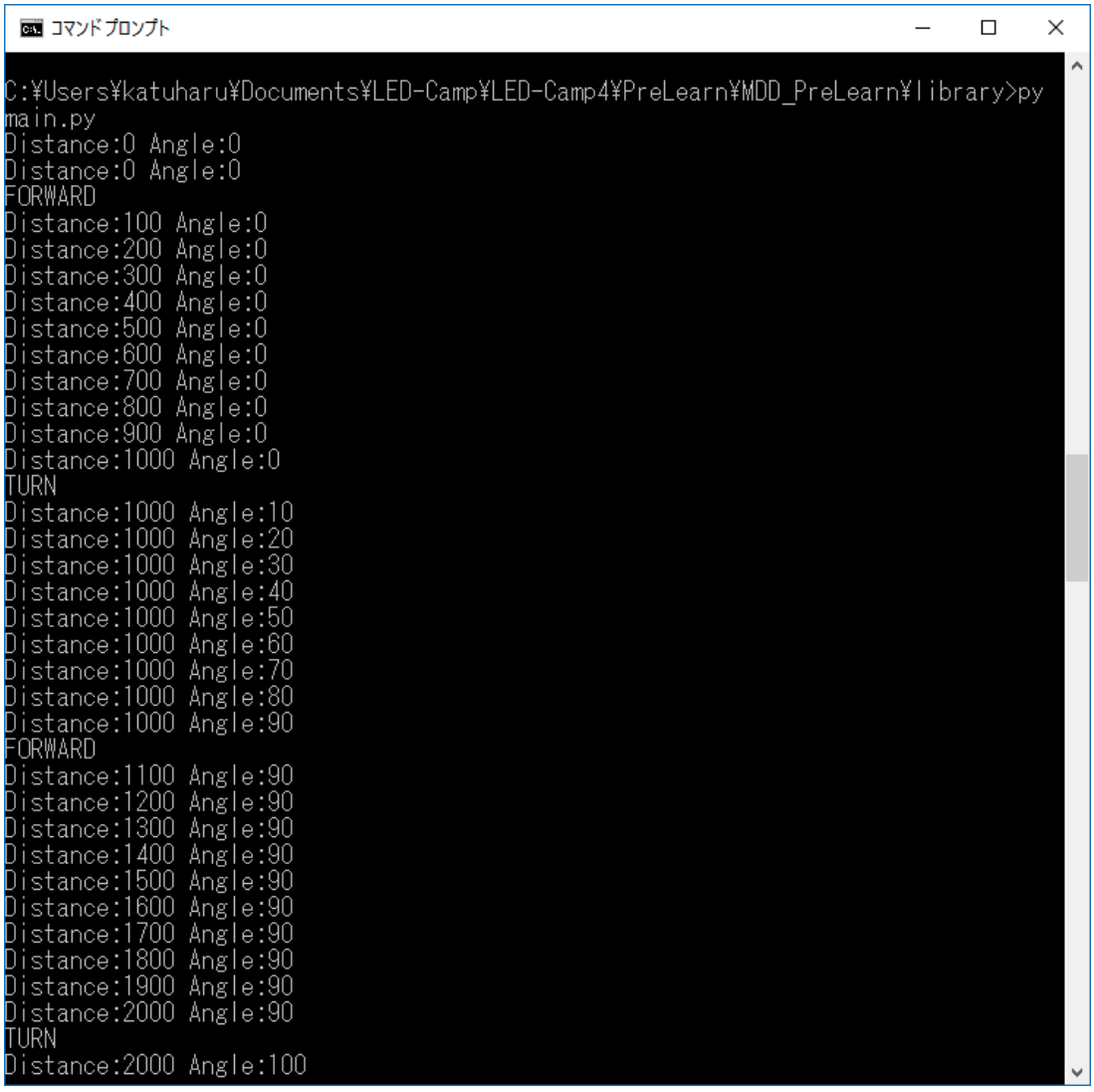
7 実行しよう

生成ができたら実行してみましょう。

コマンドプロンプトを開き、生成したディレクトリへ移動します。



Naoki Okayama

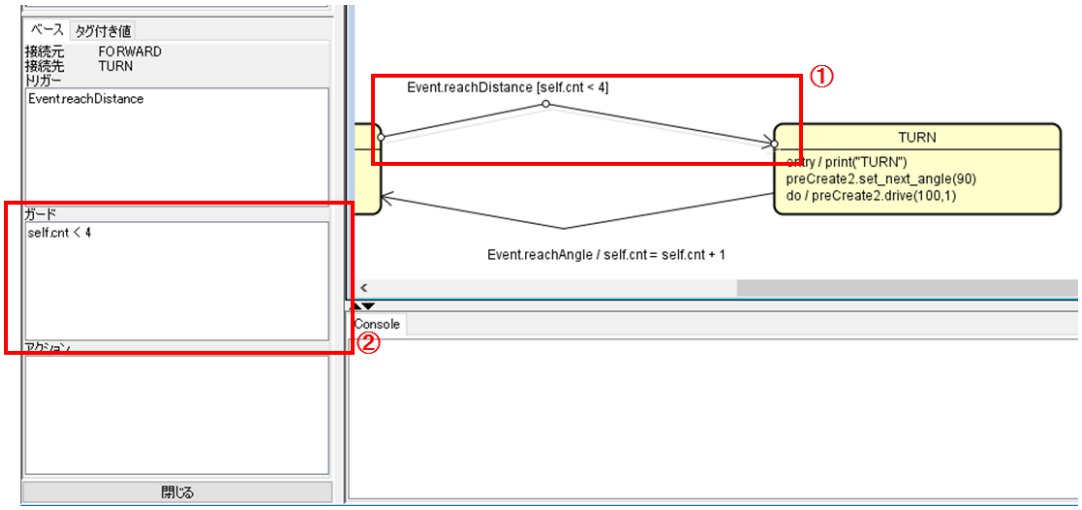


「py main.py」と打ち込み、実行してください。ライブラリ側で、現在の距離「Distance」と現在の角度「Angle」を表示しています。これまで作ってもらったモデルで、状態の入場動作として状態名の表示を行っています。はじめ「FORWARD」状態へ入り、前進。1000 まで進むと「TURN」状態へ移り 90 度まで回転。その後「FORWARD」状態へ入り・・・と繰り返している様子が見えると思います。終了させる場合は「Ctrl + C」を押して中断してください。もし、途中で「bump ○○」と表示された場合も「Ctrl+C」を押して中断してください。「bump ○○」と表示された場合は、何かに接触しライブラリ側で走行を停止した状態になります。こうなると前進も回転も行わないため、これ以上状態遷移が発生しません（ランダムで bump が発生するようになっています）。衝突の判定は、左、真ん中、右があり、それぞれ「Event.pushBumperLeft」、「Event.pushBumperCenter」、「Event.pushBumperRight」のイベントが発生します。これから、これらのイベントによる遷移を追加し、停止しないように改良を行います。

8 改良しよう

8.1 1周したら止める

今のままでは、部屋の角をぐるぐる回るようになったのは良いのですが、いつまでも回り続けてしまいます。一周したら部屋の 4 隅の掃除は終わっているので、ひとまず止まってもらいましょう。これには、ガード条件を使用します。

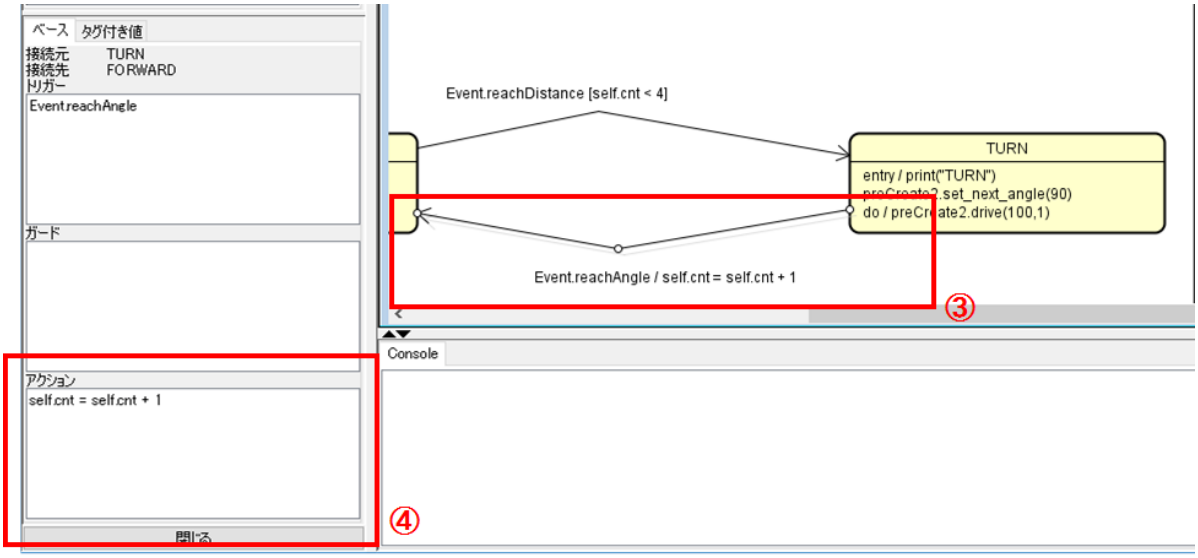


① 「FORWARD」→「TURN」の遷移を選択します

② 今回は「self.cnt」というメンバ変数に回転した回数をカウントして、4 回以上となっていた場合は TURN へ遷移をしないようにします。ですので、「FORWARD」→「TURN」の遷移のガード条件を「self.cnt < 4」と設定します。

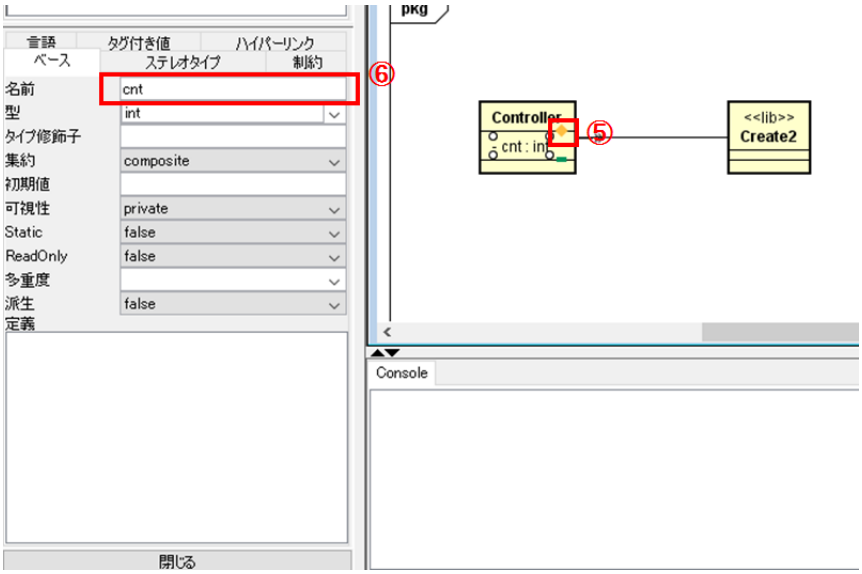
※cnt は、Controller クラスのメンバ変数のため「self.cnt」と記述します。詳細は、Python チュートリアルを参照してください。

Naoki Okayama



③ ガード条件を追加しました。あとはカウントしなければいけません。カウントは「TURN」→「FORWARD」へ遷移するときにアクションでカウントします。「TURN」→「FORWARD」の遷移を選択します

④ 左のビューのアクションを「self.cnt = self.cnt + 1」とします



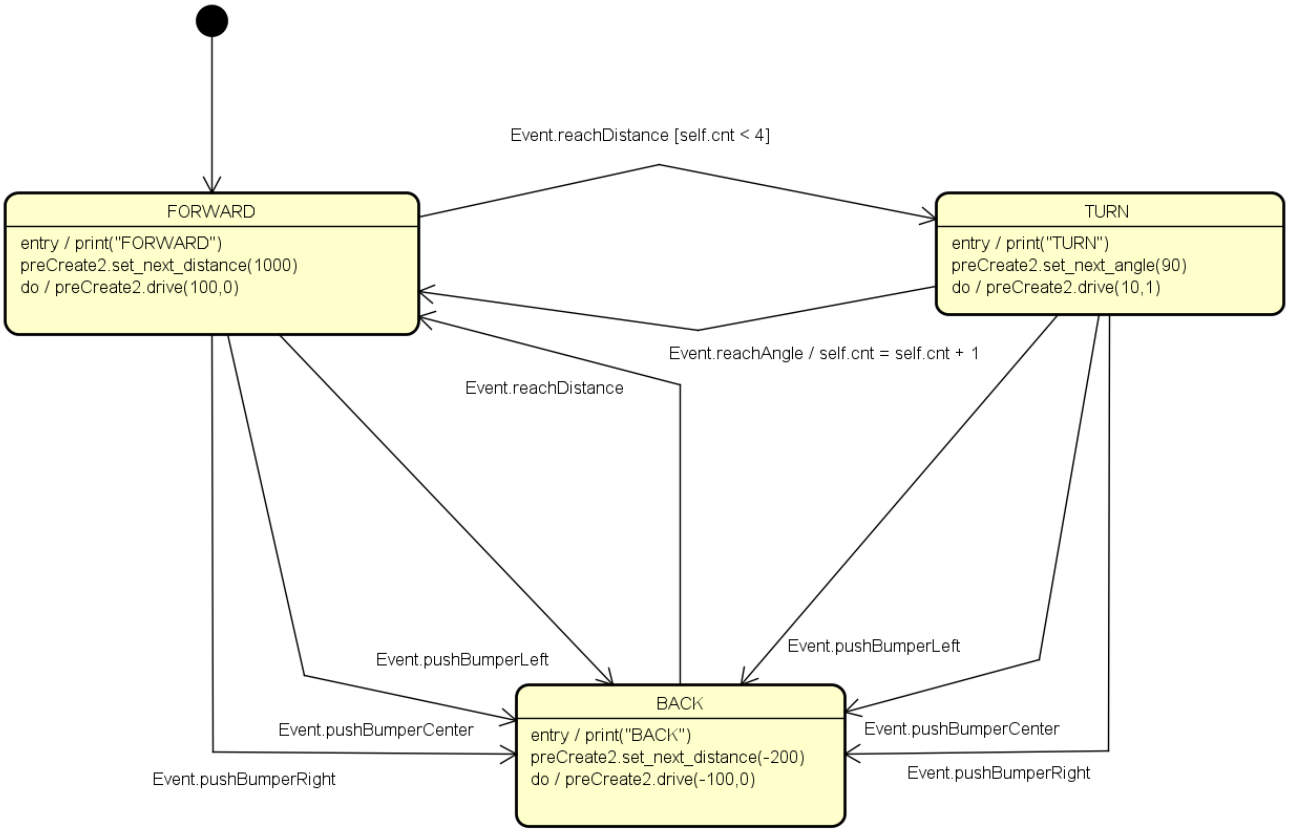
⑤ 次に Controller クラスに cnt のメンバ変数を追加します。Controller クラスの黄色のひし形をクリックします。

⑥ Controller クラスの部分や、左のビューに追加するメンバ変数の名前を設定できるので cnt とします。Python では型は自動で決定するので、設定は不要です。しかし実際には cnt は int 型の変数として定義をしたいためここでは int 型としておきます。

ここまで修正できたら、再度コード生成して実行してください。実行すると、4 回回転したあとは、どれだけ前進しても次の TURN 状態へ遷移しないことがわかります（しかし、止まってはくれません・・・しっかりと停止させる方法を当日までに考えてみてください）。

8.2　衝突しても停止させない

衝突を起こしてしまうと、PreCreate2 が停止してしまいます。今回はこれを修正していきます。障害物は、バックを行うとその間になくなるものとします。ここでは、「Event.pushBumperLeft」、「Event.pushBumperCenter」、「Event.pushBumperRight」が発生したら「BACK」状態へ遷移します。「BACK」状態では、「preCreate2.drive(-100,0)」で動作し、「-200」の距離を移動します。下図のように、モデルを修正して、コード生成を行い、実行してみてください。



9 最後に

無事、改良して動作したでしょうか？動かしていると、1 周したら曲がらなくはなったが前進を続ける・・・、「BACK」状態のときに衝突するとやはり停止してしまう・・・などのことに気が付くかと思います。もし時間があるようでしたら、こちらの対策も考えてみてはいかがでしょうか。

本番では、モデルの作成からコード生成を何度も繰り返すことになるかと思います。事前実習の間にスムーズに行えるよう頑張ってください。

10. 理解度チェック

1. クラス図を配置できますか？
2. クラス間の関連を接続できますか？
3. ステートマシン図に開始疑似状態を配置できますか？
4. ステートマシン図に状態を配置できますか？
5. 状態の動作（入場/実行/退場）を記述できますか？
6. 状態の遷移（トリガ/ガード/アクション）を記述できますか？
7. 作成したモデルからソースコードを生成できますか？
8. 生成したソースコードを実行できますか？