C#言語における静的文書生成手法の開発

学生証番号 544188　　ネットワークメディア学科 植田 哲司 （荻原研究室）

1 はじめに

プログラムに含まれる手続きを列挙し、その説明を記述した API ドキュメント(以下ドキュメント)は保守やライブラリとして提供する際に非常に有用な資料となる。そのため、多くの言語ではソースコードから自動でドキュメントを生成する機構を備えている。C#言語でもドキュメントの生成ができるが、XML 形式で出力されるためそのままでは人が見ることには適していない。そこで、今回 C#言語に焦点を当てて、ソースコードから静的なドキュメントを作成する手法を開発する。

2 モデル

今回作成するプログラムは、既存の C#言語処理系が出力する XML ドキュメント(以下 XmlDoc)を解析し、必要に応じてソースコードの情報を付加することで HTML ドキュメントを生成する。

具体的には C#コンパイラが生成する Xml ドキュメントを解析し、木構造へ変換する。ただし、このままでは構文に関する情報が不足しているため、ソースコードの解析を行って情報を補完する。最後に得られた情報を基に HTML ドキュメントを生成する。なお、この際に C#上にハードコードで HTML を記述するのではなく、テンプレートエンジンを用いて既存のテンプレートに埋め込む形で HTML ドキュメントを生成する。

3 結果

結果としては"C#のドキュメントを HTML 文書として生成する"という目的を果たせた。実際に当プログラムのソースコードやその他プログラムの解析を行い、Windows や macOS、iOS 上での閲覧を確認できた。また、Web サーバ上にそのままアップロードしても動作するため、一般公開することも容易である。

4 評価

ドキュメントは開発するプログラムの機能の概要を理解する上で非常に重要な要素となる。特に静的な HTML 形式で生成することでどのような端末でもウェブブラウザさえあれば誰でも閲覧でき、そのままウェブサーバへ配置すれば容易に全世界へ公開することができる。また、テンプレートエンジンの採用により独自の形式でドキュメントを生成することも可能で独自の形式での出力も可能である。

そのため、様々な場面で活用することができるため、非常に有用であると考えられる。

5 まとめ

ドキュメントは非常に有用な資料であるが、C#言語においては XML 形式のみで生成されるため、人が見ることに適したドキュメントが生成できない。

そこでこの XmlDoc を解析し、ウェブブラウザさえあれば誰でも閲覧することのできる HTML 形式のドキュメントを生成する。しかしながら、この XmlDoc だけでは情報が不足するため、ソースコードを解析することで情報を補完し、HTML ドキュメントを生成する。

結果、"C#のドキュメントを HTML 文書として生成する"という目的を果たし、実際に Windows や macOS、iOS 上で動作することを確認できた。更にはウェブサーバへ配置することでそのまま公開することもできた。また、HTML フォーマットを独自に作成することも可能であり、様々な場面で活用することができる。そのため非常に有用なプログラムであると考えられる。

参考文献

[1] Microsoft, XML ドキュメント コメント (C# プログラミング ガイド), https://docs.microsoft.com/ja-jp/dotnet/csharp/programming-guide/xmldoc/xml-documentation-comments

時間経過をレイヤとして記述するプログラミング手法の提案

学生証番号 544197　　インテリジェントシステム学科 上森 康太郎（荻原研究室）

1 はじめに

物語の内容が難しい作品は、以前のストーリーを思い出せなかったり、人物の関係や振る舞いを忘れてしまうことなどがある。そこで、時間経過によって変化していく事柄をうまく表現できるような形式的な記述方法を定義できないかと考えた。

2 目的

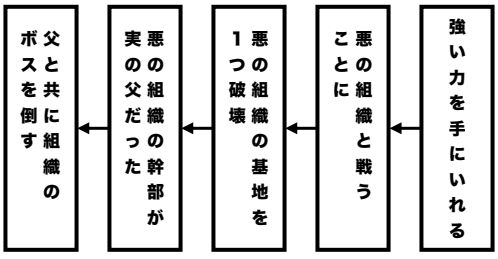
物語を記述する方法には自然言語や図が用いられることがほとんどである。前者は自由に記述できる反面、形式的でなく読み手によって解釈が変わる可能性がある。後者は視覚的な情報が多く理解しやすいが、年表であれば物事の詳細、相関図であれば時間など表現できない情報が出てくる。

形式的な記述ができれば情報の管理がしやすくコンピュータで扱いやすくなる。そうすれば情報の表示方法や情報量を簡単に変更することができ、両方の良さを生かせるのではないか。

今回は、ある時点での登場人物の状態や人物間の関係など特定の情報を取り出すこと、時間的に先行する時点に置ける情報を次の時点にも自動的に引き継ぐことの２点を目標に掲げて取り組んだ。

3 提案手法の説明

物語の形式的な記述方法を考える。まず、物語を「場面」の集合とする。「場面」とは特定の時点での登場人物などの状況や関係性の変化などを記述しているものであり、「場面」は時間的な前後関係の情報も持つ。



場面の例

前の「場面」で定義された属性は次の「場面」へ引き継ぐようにする。これは、古い「場面」から新しい「場面」を重ね合わせていくと考えることができ、すなわちそれはレイヤとみなすことができる。このようにして時間経過を記述することにした。

具体的な実装はpythonを用いたライブラリとした。場面をキーとしたデータベースとして、登場人物などの属性、場面の前後関係、登場人物同士の関係の３つを記述するようにした。登場人物や組織などはそれぞれクラスを作成し、そのインスタンスとして扱うことでデータとした。

4 議論

物語に関する特定の情報をある時点ごとに取り出すことができた。しかし、記述できるものはクラスのインスタンスのうち変数のみであり少ない情報しか記述できない。使い道としては物語を読み解く際はもちろん構成する際にも役立つのではないだろうか。

5 まとめ

物語など、時間経過をレイヤとしてプログラムを記述することで表現しやすくなる事柄があると考え、時間経過により状況が変化する事柄を表現するためのライブラリを作成した。ライブラリとしては限られた条件の情報しか記録することはできないが使用用途はあるのではないか。この研究を発展させるなら、自然言語で記述された物語を読み込んで形式的な記述にするなどしたい。

参考文献

[1] 紙名哲生, “文脈指向プログラミングの要素技術と展望”, J-STAGE, 31巻, 1号, p. 1_3-1_13 (2014).

[2] Cygames Engineers’ Blog, “階層構造によるゲーム内時間管理”,http://tech.cygames.co.jp/archives/2961/(2016).

2次元グリッドマップ上の最短経路探索アルゴリズムの比較検討

学生証番号 544430　　コンピュータサイエンス学科　岸 慎悟　(荻原研究室)

1 はじめに

カーナビで目的地までの経路を調べたり、ゲームなどのNPCを目的地まで動かすために最短経路探索アルゴリズムが利用される。それら2次元グリッドマップ上での探索に使われる代表的なアルゴリズムにA*とJPS(Jump Point Seach)がある。本研究では様々なグリッドマップに対してこれらを適用し、実行効率の比較を行った。

2 概要

A*[1]とはダイクストラ法を推定値付きの場合に一般化したものでマップ全域を探索しなくてもよいように計算方法を工夫している。

JPS[2]はA*アルゴリズムを最適化した物である。A*などの隣接地点を1つずつ移動して調べて行くのではなくJump Pointと呼ばれる分かれ道となるノードだけを保存しJump Pointの間にあるノードをカットすることで効率をあげる。

3 計算コストの比較

A*とJPSの目標地点を発見するまでの実行時間を比較するために障害物がない200×200のグリッド数のグリッドマップと200×200のグリッド数の様々な迷路型のグリッドマップで計測を行う。障害物がないグリッドマップの実行時間の結果を図1、迷路のグリッドマップでの実行時間を図2に示す。

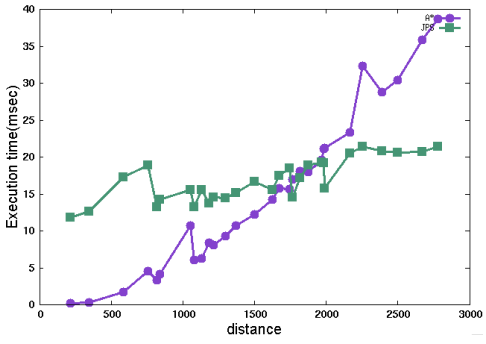


図1 200×200の障害物なしの実行時間の結果

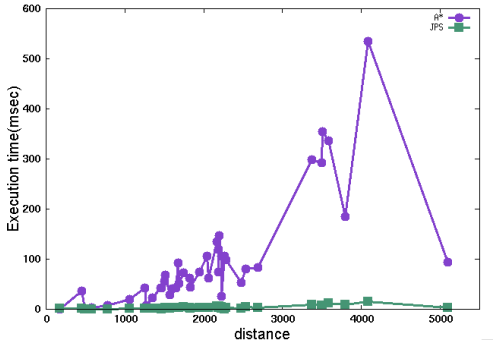


図2 200×200の迷路型の実行時間の結果

x軸は出発ノードから目標ノードまでの最短経路の長さを表しており、y軸は実行時間を表しており単位はmsecである。

4 まとめ

A*とJPSの実行時間は障害物がなく広いマップの場合、短い距離ならA*の方が良い場合もあるが、迷路型のマップの場合はJPSの方が効率が優れていることが示された。

本研究では比較実験の他に、障害物が動的に現れたり、消えたりした場合の最短経路の再計算を低減する手法についても検討を行ったが、効果的な方法は発見できなかった。この点については今後の課題としたい。

参考文献

[1] Peter E. Hart, Nils J. Nilsson, Berram Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, IEEE Transactions on Systems Science and Cybernetics vol. ssc-4 No. pp100-pp107, 1968

[2] Daniel Harabor, Alban Grastien, “Online Graph Pruning for Pathfinding on Grid Maps”, 25th National Conference on Artificial Intelligence. AAAI, 2011

状態の管理と遷移ロジックに注目した iOSアプリのアーキテクチャの提案

学生証番号 544818　コンピュータサイエンス学科　高橋 友貴　（荻原研究室）

1 はじめに

GUIを持つアプリケーションの開発にMVCアーキテクチャが長く利用され、iOSアプリの開発にも用いられる。しかしiOSアプリは多くの状態を持つことが多く、画面遷移の記述は独特なものであるため、MVCに代わるiOSアプリ開発に適したアーキテクチャが必要であると考えた。

本研究ではJavaScriptで提案されているReduxアーキテクチャ[1]とiOSアプリ開発で提案されているCoordinatorパターン[2]を組み合わせる手法を提案した上で初学者にも開発しやすい仕組みづくりを行う。

2 ReduxとCoordinator

ReduxはReact.jsで提案されているアーキテクチャであり、Webフロントエンドにおいて多くの変化し続ける状態の管理に注目したアーキテクチャである。

CoordinatorはiOSアプリ開発における複雑な画面遷移ロジックを管理しやすく、テスト容易性の向上やコードの整理に役立つとされているデザインパターンである。

3 提案するアーキテクチャ

図1は本研究で提案するアーキテクチャの概要図である。Reduxにおける Actionを状態の変更を行うMutateAction、画面遷移を行うTransitionActionが分離されている。このようにすることでReduxにおけるActionの意味の混同を防いでいる。その他に、Reduxで状態変更を行うReducerと同等にFlowResolverという画面遷移ロジックを記述する機構をCoordinatorを参考に用いていることでViewから直接画面遷移を指示することなく、遷移Actionとして発行してReduxにより近づけている。

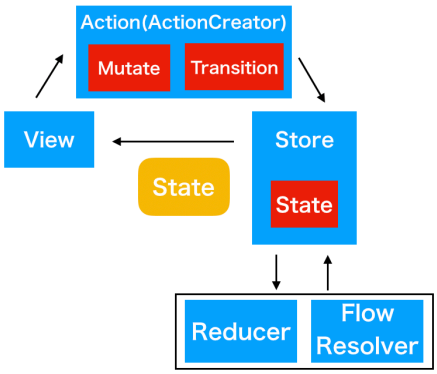


図1 提案するアーキテクチャの概要図

4 議論

iOSアプリにおける画面遷移の複雑さをCoo-rdinatorパターンによって集約し、状態の変更と監視を明確にすることで予期せぬ状態変更に対する対処がしやすく、Actionによる変更で状態がどのようになるかが明確になったことでテストが容易になる利点などがある。逆にReduxに過度の束縛をしてしまったことにより初学者にはReduxの理解が求められ、開発しやすくなったとは言い難いと考えられる。

5 まとめ

本研究ではReduxとCoordinatorを用いたアーキテクチャの実装から、それを利用したサンプルアプリまでの実装を行なった。iOSアプリに適したアーキテクチャを考案することでViewControllerの肥大化を防いだり、MVCでは難しいと考えた状態の管理のしやすさに対して多くの利点があった。

参考文献

[1] Redux, https://redux.js.org/introduction/motivation

[2] Will Townsend 2016-12-18, https://will.townsend.io/2016/an-ios-coordinator-pattern