マイクロシミュレーション説明資料

B4　小川大智

目次

目次

1.マイクロシミュレーション概要

2.配布資料一覧

3.ソースコード
- a.オーバービュー
- b.RLモデル
- c.経路列挙モデル
- d.Car Accleration モデル

4.インプット

5.計算の実行・可視化
- 実行環境の整備
  - Mac
  - Windows
  - 参考
- パスの設定
- 計算
  - Mac
  - Windows
- 可視化

6.課題の説明

1.マイクロシミュレーション概要

今回紹介するマイクロシミュレーションは、与えられた交通の需要から自動車、バス、鉄道、歩行者といったエージェントを生成し、それらをある決まりに従って移動させていくことで、道路の混雑状況や交通量などを見ることができるものです。

均衡配分では幹線道路レベルでの交通量を見ることができますが、マイクロシミュレーションでは、信号や車両間の相互作用などを考慮することや経路列挙とRLモデルを用いた現実に寄せたモデル化をすることでより細かいレベルでの評価をすることができると考えられます（例えば、駅のまわりの整備の効果など）。一方で、たくさんのエージェントを発生させるほど、また、一つのエージェントの挙動を計算するための負荷が大きいほど、計算資源を多く必要とすることになります。比較的最近になってこのようなシミュレーションが可能となってきたことも、ムーアの法則に従った計算機の高性能化によるものですが、現時点で利用可能な計算資源に限りがあることからも、モデル化による簡略化が必要不

可欠です。どのような規模感で何を見るのかによって、モデルを簡略化したり、逆に精緻化したりといった選択が必要になる場合もありますが、今回の演習の中では都市の計画と並行してこれを行うということに醍醐味があるといえるかもしれません。

2.配布資料一覧

配布資料は以下の通りです。

- マイクロシミュレーション説明資料：このドキュメント
- Hongo
    - src：シミュレーションのソースコード
    - input：シミュレーションのインプットデータ
    - output：計算結果を入れるフォルダ（今は空）
    - CMakeLists.txt：CMakeを使うときのビルド関連の情報（Macではほぼ使わない）
- InputCreator：readme.txt記載の通り

3.ソースコード

配布するソースコードとアルゴリズムの概要の説明です。読まなくとも計算を実行することはできますが、何をやっているかのイメージをつかむために一読してから計算を行ったほうが良いと思われます。

a.オーバービュー

使われている主なクラスの説明です。c++では、機能をクラス化することでソースコードの保守性を高めています。

- Simulation：シミュレーション本体です。タイムステップ（1秒）ごとのエージェントの位置計算と計算結果の書き出しを行います。
- Vehicle：車両の挙動を計算します。周辺の他の車両や前方の信号などの把握や車線変更の可否の判定、Car Accelerationモデルに基づく車両の加速度、速度の更新を行います。
- LaneChoiceSet：各エージェントのODから、経路列挙のアルゴリズムで通りうる車線を列挙します。
- LaneChoice：RLモデルに基づいて、列挙された車線の中から行先の車線を選択します。
- Bus：インプットに含まれるバスの供給データから、バスの経路やバス停の出発時間などを管理します。
- Agent：歩行者とほぼ同じ意味です。RouteChoiceクラスでRLモデルによる経路選択をしながら移動します。

b.RLモデル

RLモデルは、各選択肢の将来の期待効用をもとに、逐次的に次のリンクを選択するモデルです。このシミュレーションでは、時間割引率βを用いて将来の効用を

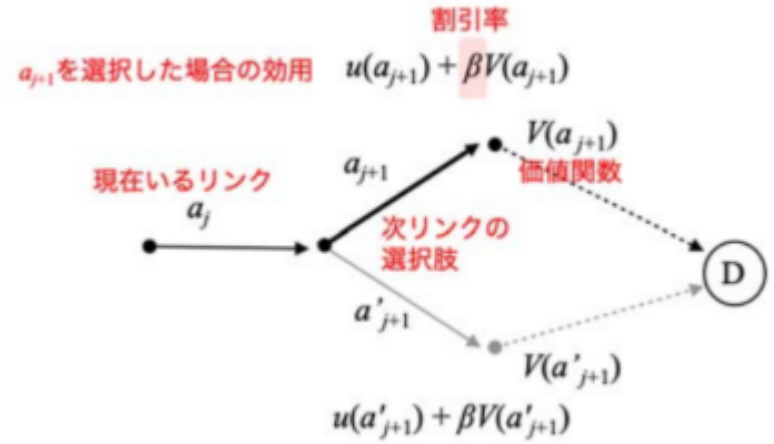
割り引くことで、行動の計画的でなさ（目的地まですべての経路を決めてから家を出ることはあまりない）を考慮しつつ経路選択を行っています。

現在いるリンクをa_jとするとき、a_jからa_{j+1}へ移動する場合の効用$U(a_{j+1})$は、

$$U(a_{j+1}) = u(a_{j+1}|a_j) + \beta V(a_{j+1}) = v(a_{j+1}|a_j) + \beta V(a_{j+1}) + \mu\varepsilon(a_{j+1})$$

$$V(a_j) = E[\max_{a_{j+1} \in A(a_j)} \{v(a_{j+1}|a_j) + \beta V(a_{j+1}) + \mu\varepsilon(a_{j+1})\}]$$

ただし、$u(a_{j+1}|a_j)$はa_jからa_{j+1}へ移動したときのa_{j+1}の効用、$v(a_{j+1}|a_j)$はその確定項、$\varepsilon(a_{j+1})$はその誤差項、$V(a_{j+1})$はa_{j+1}の将来期待効用（価値関数）、βは時間割引率（0より大きく1以下）、μはスケールパラメータ。



MNL（multinominal logit）と同様にして、リンクa_{j+1}が選択される確率$p(a_{j+1}|a_j)$は、

$$p(a_{j+1}|a_j) = \frac{exp(\frac{1}{\mu}\{v(a_{j+1}|a_j)+\beta V(a_{j+1})\})}{\sum_{a'_{j+1} \in A(a_j)} exp(\frac{1}{\mu}\{v(a'_{j+1}|a_j)+\beta V(a'_{j+1})\})}$$

ただし、$A(a_j)$はa_jの下流リンクの集合である。

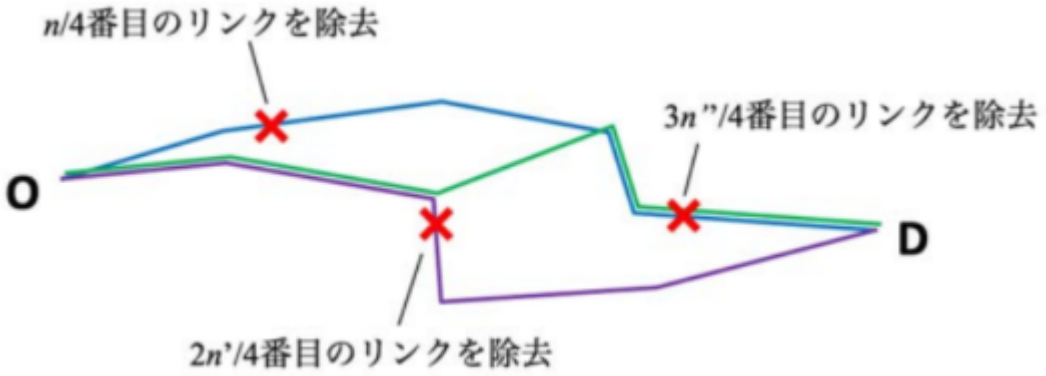
※参考　一般化RLモデルを用いた災害時の経路選択行動分析

c.経路列挙モデル

経路選択の計算を行う際にネットワーク全体を用いると計算量が大きくなるため、あらかじめリンクの集合を限定し、その中から経路選択を行うこととします。対象となるリンク集合は以下のようなアルゴリズムを用いて生成します。

1. あらかじめ与えられたODに対して、最短経路探索アルゴリズム（今回は負の場合は存在しないため、AStarアルゴリズムやダイクストラ法など）を用いて最短経路を探索する。このとき、コストとしてリンク長またはリンク旅行時間を選択する。この経路のリンク数をnとする。この経路上のリンクをリンク集合に加える。
2. ネットワークから1.の経路の4/n番目のリンクを除き、そのネットワークで最短経路探索を行う。最短経路が見つかった場合はその経路上のリンクをリンク集合に加える。この経路のリンク数をnとする。最短経路が見つからない場合はリンクを戻し、3.に進む。

3. 2.と同様にネットワークから2n/4番目のリンクを除き、最短経路探索を行う。最短経路が見つかった場合はその経路上のリンクをリンク集合に加える。この経路のリンク数をnとする。最短経路が見つからない場合はリンクを戻し、4.に進む。
4. 3.と同様に3n/4番目のリンクを除き、最短経路探索を行う。最短経路が見つかった場合はその経路上のリンクをリンク集合に加える。この経路のリンク数をnとする。最短経路が見つからない場合はリンクを戻し、5.に進む。
5. 2.にもどり、指定された経路数に達するまでこれを繰り返す。



d.Car Accleration モデル

このシミュレーションでは、車両（自動車やバス）の加速度を1秒ごとに更新しながら、車両の位置を計算していきます。

車両は、前方のすぐ近くに他の車両がある場合にはその車両に追従するような動きをしますが、そうでない場合は制限速度近くまで加速していきます。車両がどちらの挙動をするかは車頭時間によって以下のような式で判別されます。ただし、$P(h_n(t) \le h_n^*)$ は車両が追従モデルに入る確率です。

$$P(h_n(t) \le h_n^*) = \begin{cases} 1 & h_n(t) \le h_{min}^* \\ 1 - \dfrac{\Phi\left(\frac{h_n(t)-\mu_h}{\sigma_h}\right) - \Phi\left(\frac{h_{min}^*-\mu_h}{\sigma_h}\right)}{\Phi\left(\frac{h_{max}-\mu_h}{\sigma_h}\right) - \Phi\left(\frac{h_{min}^*-\mu_h}{\sigma_h}\right)} & h_{min}^* \le h_n(t) \le h_{max}^* \\ 0 & otherwise \end{cases}$$

ただし、$h_n(t)$は車両nの前方車両との車頭時間、h_{min}^*とh_{max}^*は車頭時間の閾値の最小値と最大値、μ_hとσ_hはh_n^*の従う両端切断正規分布の平均と分散。

追従モデルの加速度は、

$$a_n^g(t) = s^g[X_n^g(t)] f^g[\Delta V_n(t-\tau_n)] + \varepsilon_n^g(t)$$

ただし、gは減速 or 加速、$s^g[X_n^g(t)]$は車両nの感応度、$X_n^g(t)$は感応度の説明変数、$\Delta V_n(t-\tau_n)$は前方車両との相対速度、ε_n^gは誤差項。

自由走行モデルの加速度は、

$$a_n^{ff}(t) = s^{ff}[X_n^{ff}(t)]f^{ff}[V_n^{DS}(t-\tau_n) - V_n(t-\tau_n)] + \varepsilon_n^{ff}(t)$$

ただし、$s^{ff}[X_n^{ff}(t)]$は車両nの感応度、$X_n^{ff}(t)$は感応度の説明変数、$V_n^{DS}(t-\tau_n)$は希望速度（普通は道路の制限速度）、$V_n(t-\tau_n)$は車両の速度。

※参考 Modeling drivers' acceleration and lane changing behavior

4.インプット

シミュレーションを回すためのインプットは以下の通りです。

- rcParam.csv
  RouteChoiceに用いるパラメータ。
  type：交通モード（0：自動車、1：鉄道、2：自転車、3：歩行者、4：駅接続リンク）

- vbParam.csv
  車両挙動モデルに用いるパラメータ。
  av：自動運転か否か

- acParam2.csv
  ActivityChoiceに用いるパラメータ。今のコードでは使われていない。

- KotoCarLinks.csv
  自動車リンクデータ。
  lengthの単位はm
  travelTimeの単位は秒
  type：リンク種別（1:本線（上下非分離）リンク、2:本線（上下分離）リンク、3:連結路（本線間の渡り線）リンク、4:交差点内リンク、5:連結路（ランプ）リンク、6:本線と同一路線の側道リンク、7:SA等側線リンク、8:自転車道等リンク、9:本線側道接続リンク、10:DRMメッシュ境界リンク、0:未調査）
  velocity：制限速度。元データに未調査が多く道路種別から作成したため、実際と相違あり。
  NumOfLane：車線数
  intersection：信号のある交差点番号

- KotoCarNodes.csv
  自動車ノードデータ。
  type：交通手段番号（0：自動車）

- RoadLaneConnectivity.csv
  車線接続情報。
  upLaneID：上流車線ID。車線IDはリンクID×100+車線位置番号（一番右が1で左に行くほど1ずつ増える）

dnLaneID：上流車線から車線変更なしに行ける下流車線のID。
turn：曲がるか否か（0：直進、1：右左折Uターン）
crossing：上流リンクから繋がっているリンクが2つ以上あるか否か（0：ない、1：ある）

- KotoSignal.csv
  信号の位置と制御の情報。信号は車線ごとに割り当てる。
  id：信号ID
  laneID：信号が割り当てられた車線のID。
  Position：車線の後ろ側から見た信号の位置（単位はm）。
  offset：初期の位相のずれ。
  red：赤の長さ
  yellow：黄色の長さ
  green：青の長さ

- KotoIntersection.csv
  上流から下流へ移動する際に相互作用を考える車線の情報。例えば、右折する際の対向車線など。
  upLaneID：上流車線ID
  dnLaneID：下流車線ID
  crossLaneID：相互作用を考える車線のID
  drection：相互作用を考える車線が上流車線の前方か後方か（0：前方、1：後方）

- KotoOD.csv
  交通需要データ。
  StartWalkNodeID：出発ノードのID。ただし、自動車の場合は CarNodeID+30000000。
  EndWalkNodeID：到着ノードのID。ただし、自動車の場合は CarNodeID+30000000。
  DepartureTime：シミュレーション開始時間を0とした出発時間。単位は秒。例えば、シミュレーションが午前0時から始まっているとすると、朝6時は3600×6。
  Mode：交通手段番号（0：自動車、1：鉄道、2：自転車、3：歩行者）
  Expansion：発生トリップ数

- RouteLInks.csv
  バス路線が通過するリンクのデータ。
  RouteID：バス路線のID
  Name：バス路線の名前
  linkID：バス路線が通過するリンクのID

- RouteStations.csv
  バス路線に対応させたバス停の情報。一つのバス停でも、進行方向によって同じリンクに割り当てられるとは限らないため。
  RotueStationID：routeStationのID
  RouteStationName：routeStationの名前
  RouteID：バス路線ID
  StationID：対応するバス停のID
  LinkID：routeStationが割り当てられたリンクのID

LaneID：routeStationが割り当てられた車線のID
Position：routeStationのリンク上での位置。（リンクの始点は0、終点は1。）
Mode：交通手段（6：バス）

- Stations.csv
  バス停の位置情報。
  StationID：バス停ID
  name：バス停名

- Timetable.csv
  バスごとの時刻表情報。
  RouteID：路線ID
  upRouteStationID：DepartureTimeに対応するバス停。このバス停を発車する時刻がDepartureTime。
  dnRouteStationID：次に行くバス停
  DepartureTime：upRouteStationの発車時刻
  DurationTime：発車時刻から次のバス停の到着時間までの時間（秒）
  TrainID：バス ID
  TrainName：バス名

- Trains.csv
  バスの車両情報。
  TrainID：バス ID
  RouteID：そのバスが通る路線のID
  StartTime：そのバス車両が最初のバス停を出発する時刻
  TrainName：バス名

5.計算の実行・可視化

計算を実行から結果の可視化までの流れを説明します。

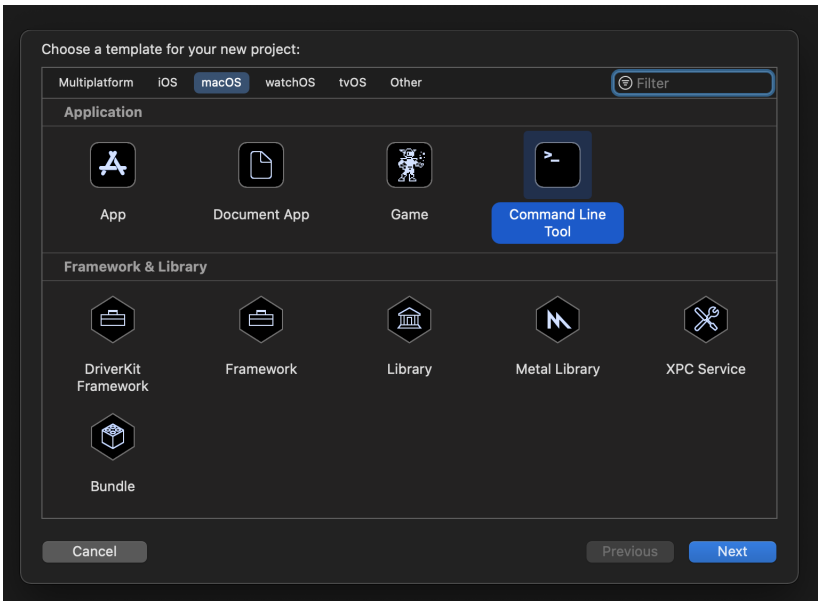
1. 実行環境の整備

既に実行環境がある人は読み飛ばして構いません。また、説明の途中で出てくるアプリケーションなども、すでにインストールされている場合は、無視して構いません。

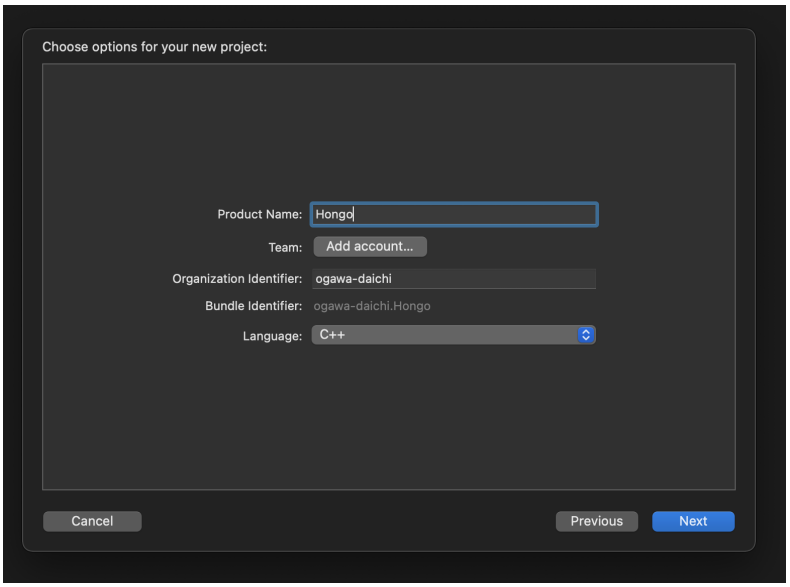
a. Mac

i. Xcodeをインストールします。AppleStoreで検索すれば出てくると思います。
ii. 「Create a new Xcode project」を押します。

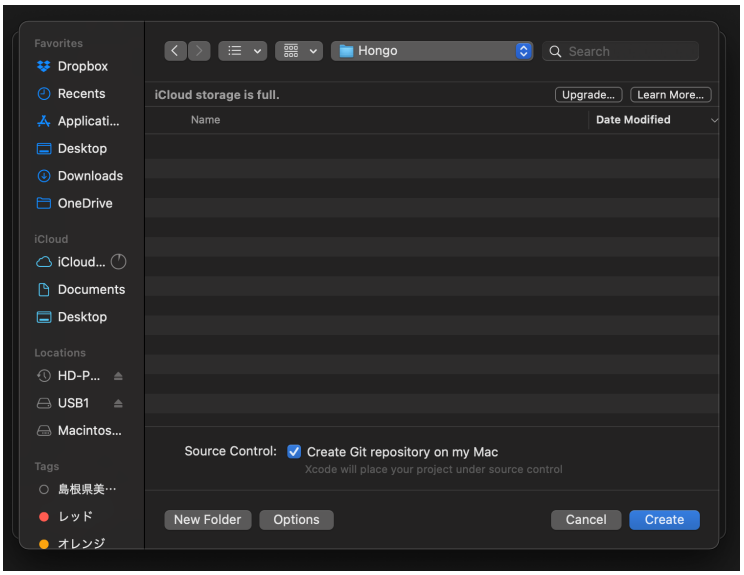
iii. macOSの「Command Line Tool」を選択してNextを押します。



iv. ProductNameを適当に記入します。Hongoなどとしておけば良いです。言語はC++にします。Nextを押します。

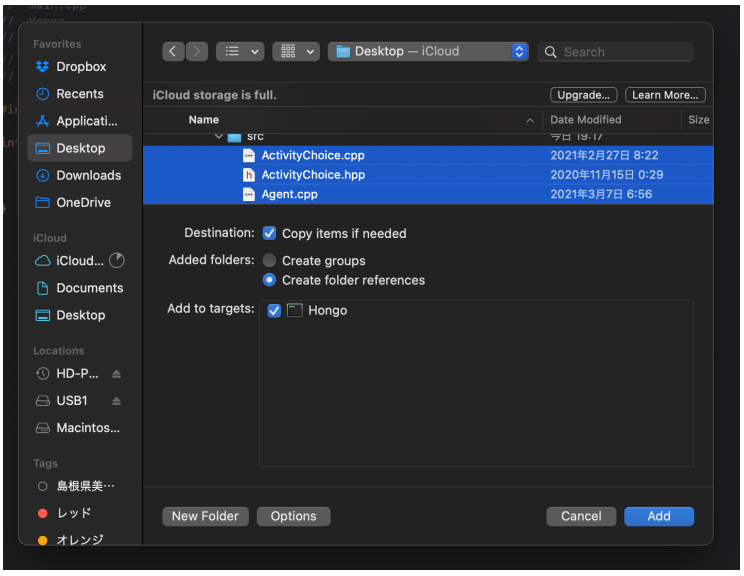


v. プロジェクトを作成するフォルダを選択します。Createを押します。



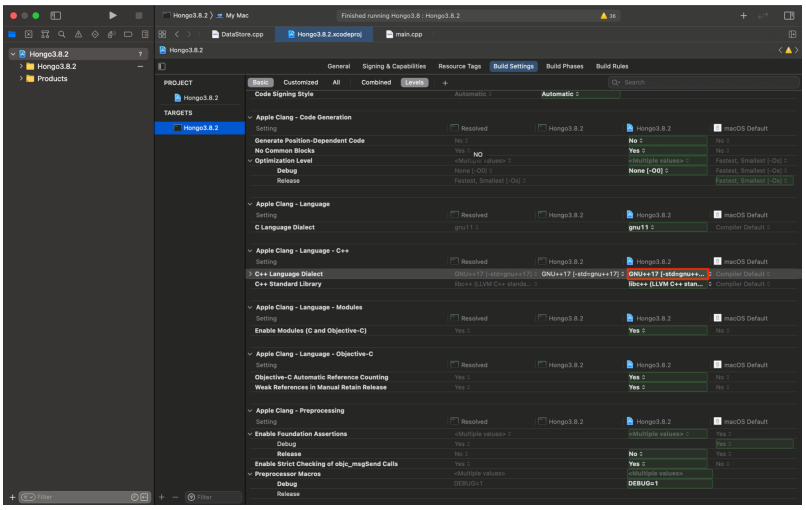
vi. No author information ... というようなエラーが出ることもありますが、無視して大丈夫です。

vii. ソースファイルを追加します。iv.でつけた名前のフォルダを選択した状態で、メニューの「File」→「Add Files to "ivでつけた名前"...」を押します。写真のようにチェックを入れた状態で、配布するsrcフォルダの中身を全て選択し、Addを押します。ファイルが追加されますが、srcフォルダに入っていなかったmain.cppファイルを消します（右クリック→「Delete」を押します）。その後に出てくるウィンドウは「Move to Trash」で構いません。



viii. （追記）XcodeではデフォルトのC++のバージョンがC++14になっているようで、これだとビルドの途中でエラーが出るようです。Xcode上で「～.xcodeproj」を開くと各設定を変えることができます。「Build Settings」→「Apple Clang - Language - C++」→「C++ Language Dialect」の行を、「C++17」か「GNU++17」に変えてくだ

さい。



以上でプログラムを実行する環境が整いました。

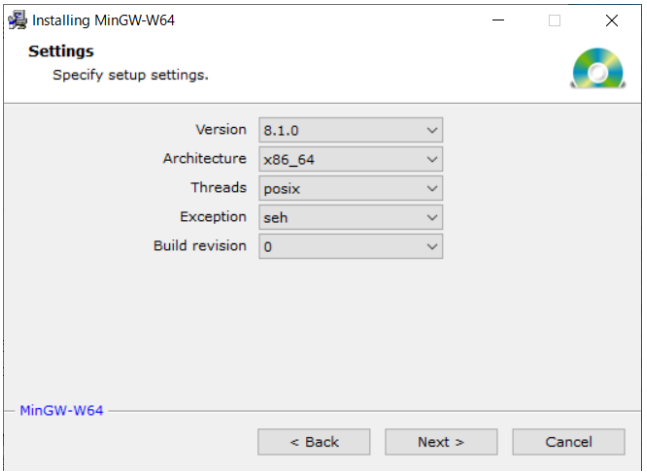
b. Windows

i. C++のコンパイラとしてGCCをインストールします。windows上でGCCを動作させるためにMinGWをインストールします。MinGWのインストーラーはこちらからダウンロードできます。

ii. ダウンロード出来たらそのファイルを実行します。発行元不明のアプリケーションの警告画面が出てきますが、実行を許可してください。

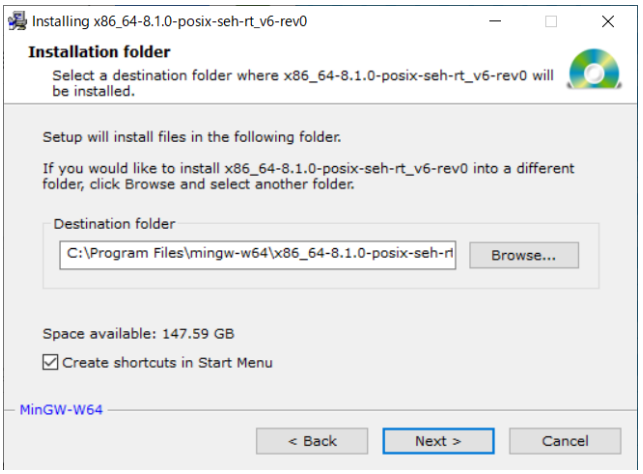
iii. Nextを押します。

iv. Architectureを確認します。Windows10が64bit版なら「x86_64」を、32bit版なら「i686」を選択します。他は変えなくて大丈夫です。64か32かは、「スタートボタン」→「設定」→「システム」→「詳細情報」→「システムの種類」で確認できます。下の場合は64bit版です。Nextを押します。



システムの種類　　64 ビット オペレーティング システム、x64 ベース プロセッサ

v. インストール先を指定します。特に理由がなければデフォルトのままで構いません。Nextを押すとインストールが始まります。



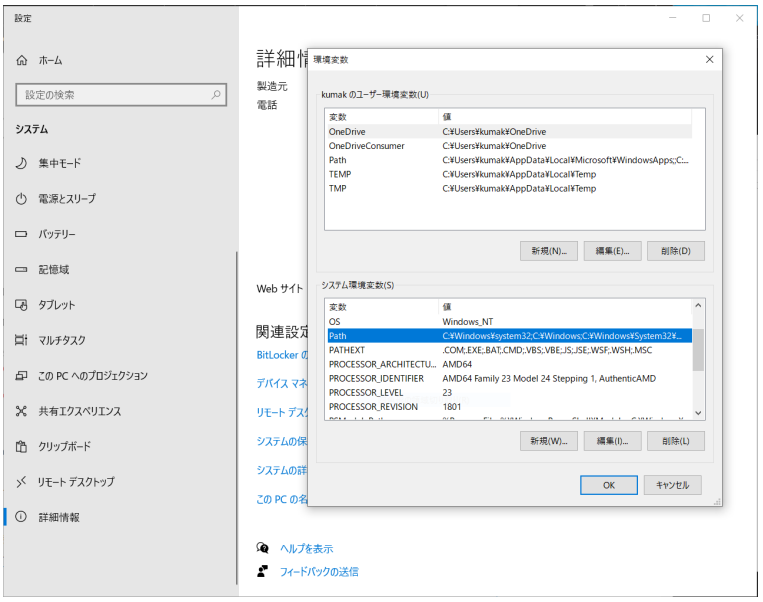
vi. NextとFinishを押してインストールは完了です。

vii. GCCがインストールできていることを確認します。「スタートメニュー」→「MinGW W64 Project」→「Run terminal」を実行します。

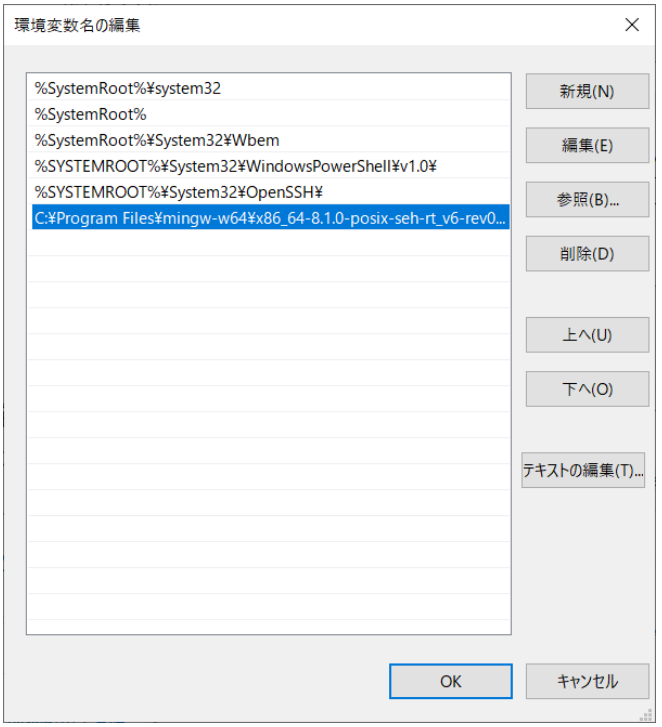
viii. ターミナルが出てくるので、「gcc -v」と入力してEnterを押します。GCCのバージョンが表示されればインストールが成功しています。

ix. 次にPowerShellからGCCを実行できるようにパスを追加します。「スタートメニュー」→「設定」→「システム」→「詳細情報」→「システムの詳細設定」→「環境変数」をクリックします。「システム環境

変数」の「Path」を選択して、「編集」を押します。



x. 「新規」を押して、GCCのパスを追加します。GCCのパスは、コマンドプロンプトで「where gcc」→Enterで出てくるものの「\gcc.exe」の前までです（つまり、「C:\~\mingw64\bin」のような感じです）。

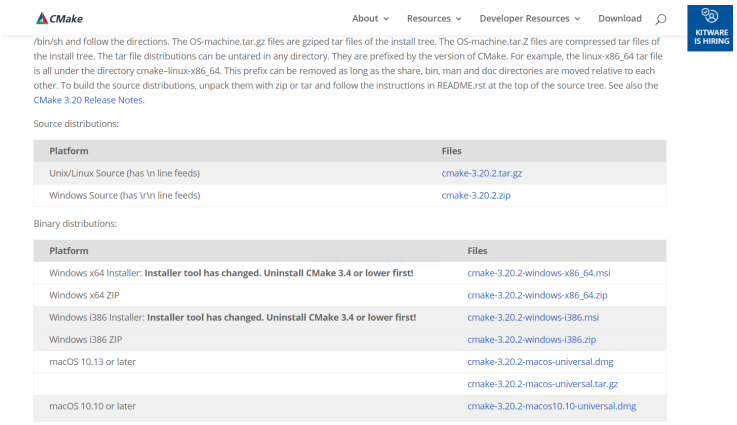


xi. 「OK」を押して、開いていたウィンドウを順に閉じます。

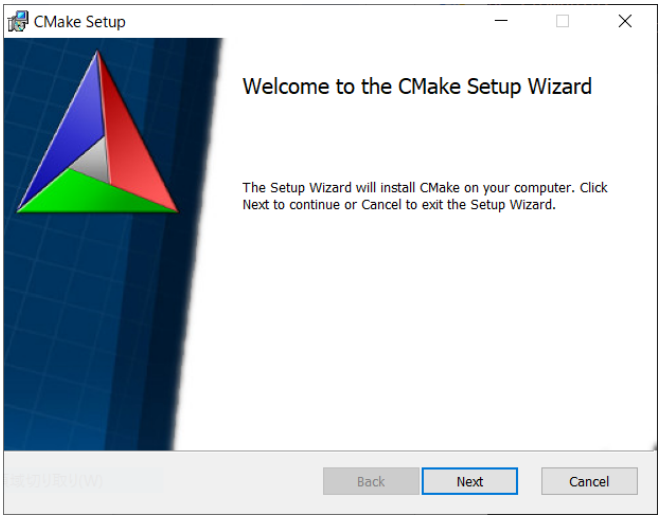
xii. Windows PowerShellを起動し、「gcc -v」と打ってEnterを押します。GCCのバージョンの情報が出てきたら設定完了です。

今回は複数のソースファイルからビルドすることになるので、それらの管理のためにCMakeを利用します。ここからCMakeのインストールなので、すでにパソコンに入っている人は読み飛ばしてください。

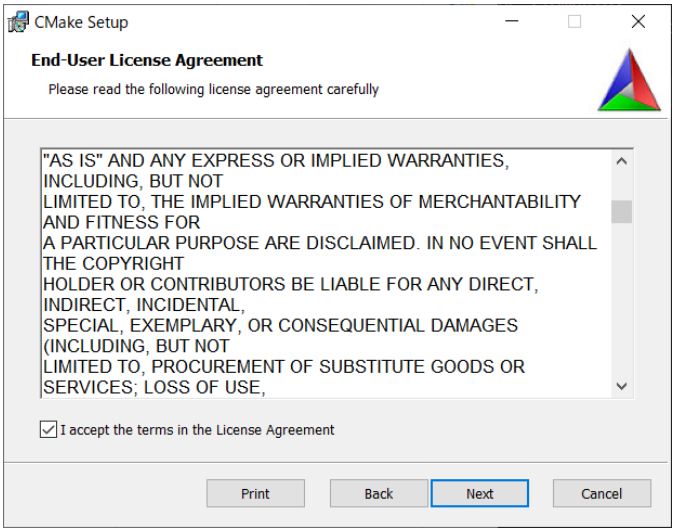
xiii. こちらから自分のパソコンにあったCMakeのインストーラーをダウンロードします。Binary Distributionの一番上か上から3番目になると思います。



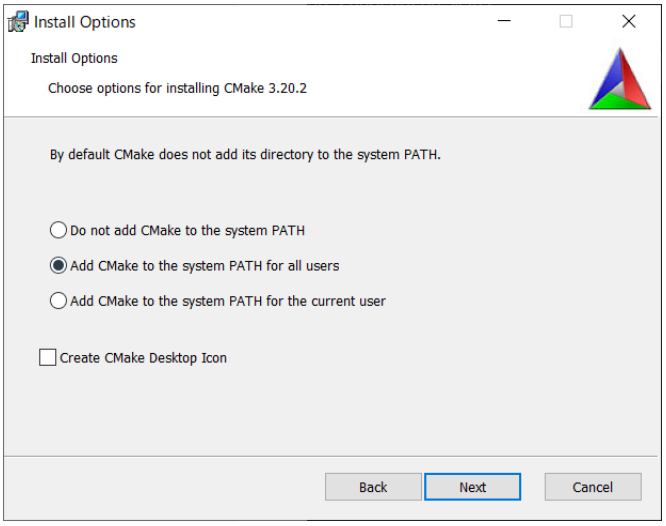
xiv. ダウンロード出来たら開いて、「Next」を押します。



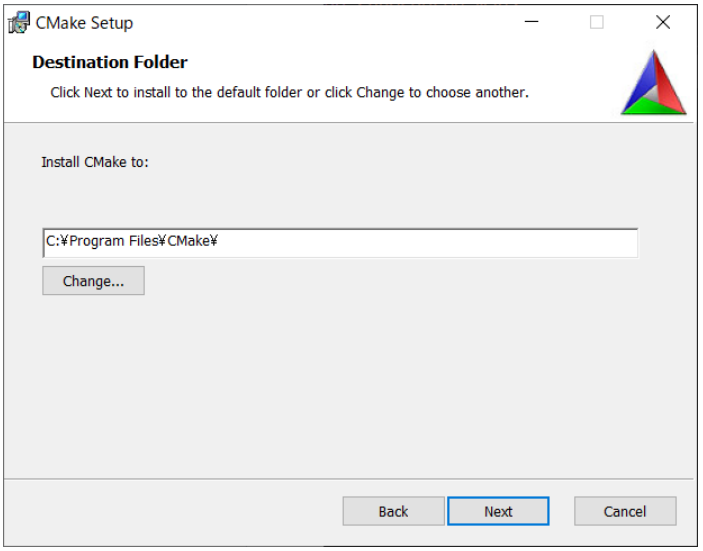
xv. ライセンス許諾を読んで同意して「Next」を押します。



xvi. 「Add CMake to the system PATH for all users」にチェックを入れて「Next」を押します。



xvii. インストール先のフォルダを選択します。特に理由がなければデフォルトのままで構いません。「Next」、「Install」を押します。「プログラムがコンピューターに変更を加えることを許可しますか…」のようなものが出てきたら、許可してください。



xviii. インストールが終わったら「Finish」を押してください。

xix. CMakeがきちんとインストールできていることを確認します。コマンドプロンプトで「where cmake」、「where cmake-gui」と打って、それぞれパスが表示されれば大丈夫です。

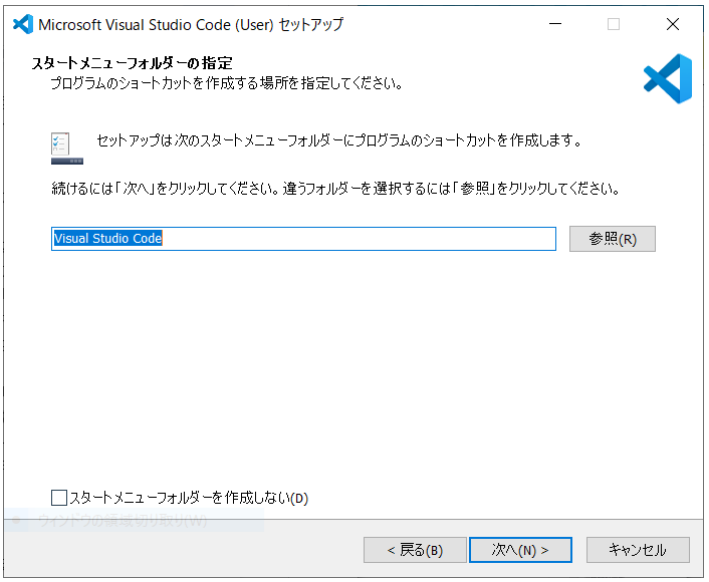
ここからエディターをインストールします。エディターはソースコードを編集するためのアプリケーションで、オートフィルやデバッグ機能などプログラムを作成するうえで便利な機能が多く実装されています。今回はVisual Studio Codeを使います。他のものがすでにインストールされている場合はそちらでも構いません。

xx. こちらのページで「Download for Windows」を押します。

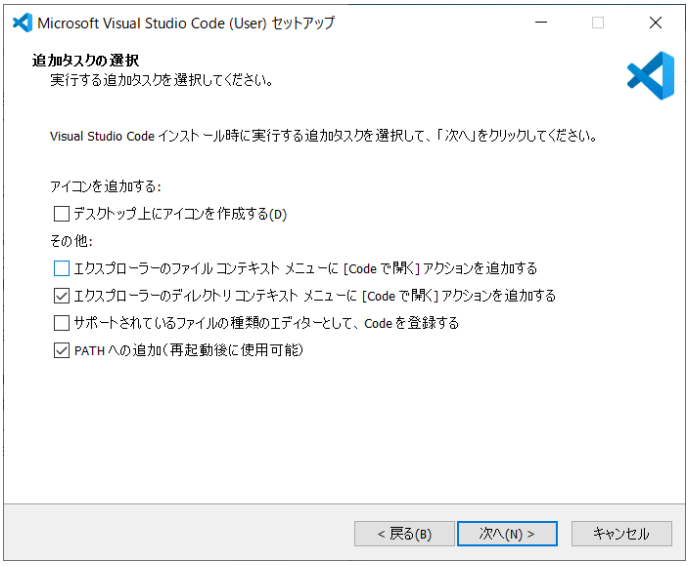
xxi. ダウンロードが完了したら、ファイルを開きます。利用許諾を確認して「同意する」を押します。インストール先を聞かれますが、特に

理由がなければデフォルトのままで構いません。「次へ」を押します。

xxii. そのまま「次へ」を押します。



xxiii. そのまま「次へ」を押します。

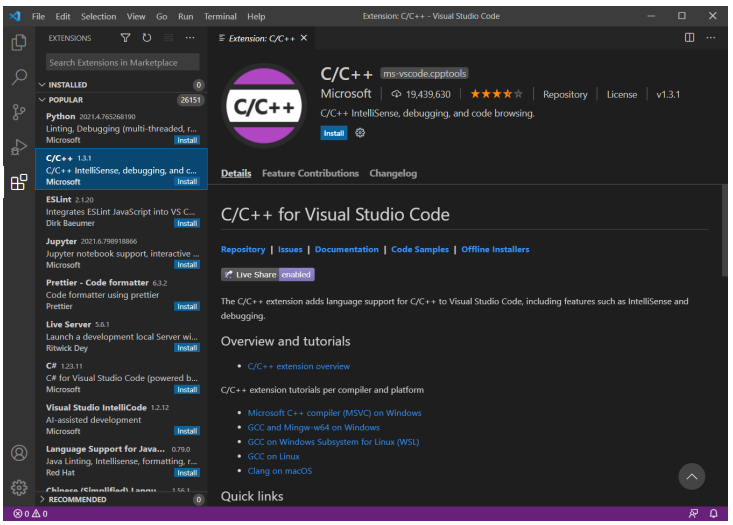


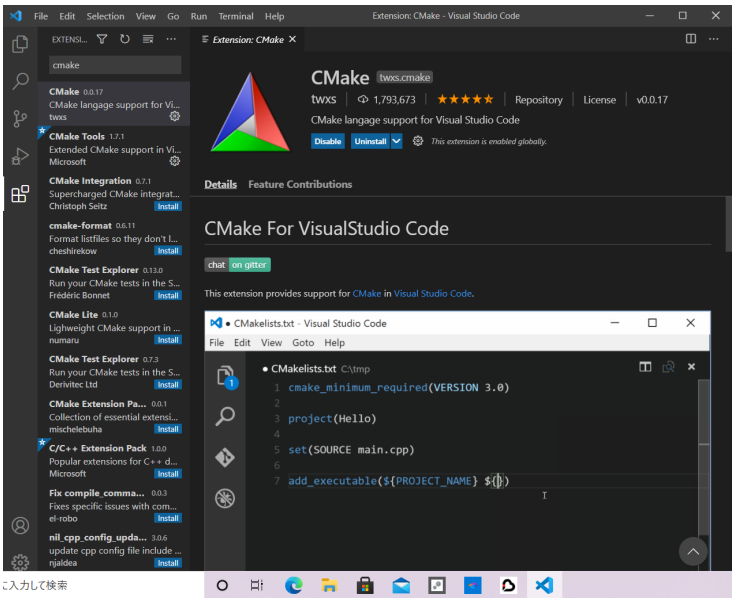
xxiv. 内容を確認して「インストール」を押します。

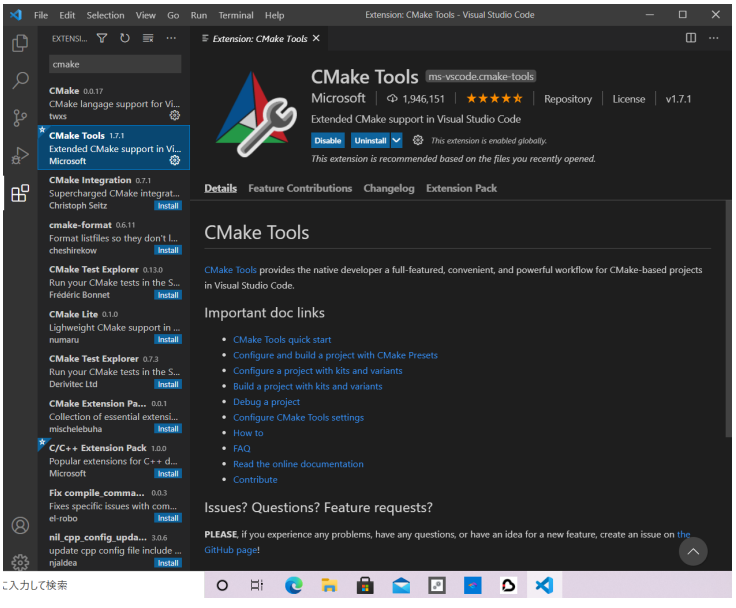
xxv. インストールが終わったら「完了」を押して終了します。

次にVisual Studio Codeを使ってC++のプログラムを実行できるように設定します。

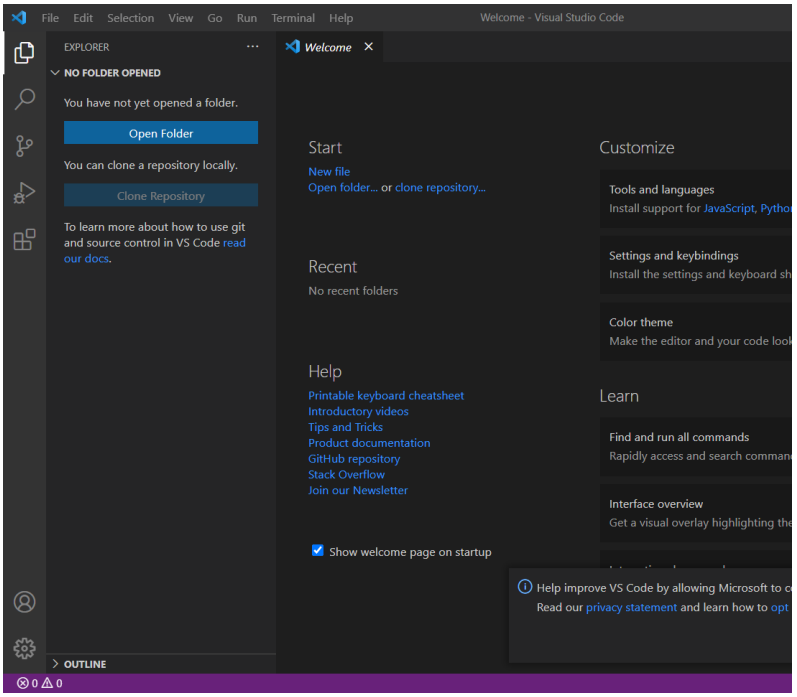
xxvi. Visual Studio Codeのウィンドウを開いて、左側のメニューの「Extensions」を押し、C/C++とCMakeのの拡張機能をそれぞれ検索します。写真のもの（全部で3つ）をインストールします。


File Edit Selection View Go Run Terminal Help
Extension: C/C++ - Visual Studio Code
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
POPULAR
Extension: C/C++
C/C++
ms-vscode.cpptools
Microsoft
19,439,630
Repository
License
v1.3.1
C/C++ IntelliSense, debugging, and code browsing.
Install
Details
Feature Contributions
Changelog
C/C++ for Visual Studio Code
Repository | Issues | Documentation | Code Samples | Offline Installers
Live Share enabled
The C/C++ extension adds language support for C/C++ to Visual Studio Code, including features such as IntelliSense and debugging.
Overview and tutorials
C/C++ extension overview
C/C++ extension tutorials per compiler and platform
Microsoft C++ compiler (MSVC) on Windows
GCC and Mingw-w64 on Windows
GCC on Windows Subsystem for Linux (WSL)
GCC on Linux
Clang on macOS
Quick links

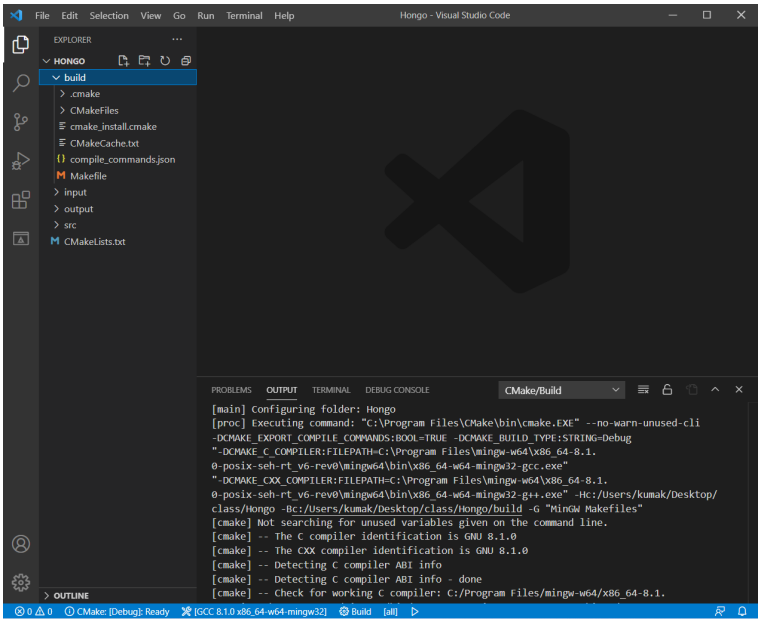

File Edit Selection View Go Run Terminal Help
Extension: CMake - Visual Studio Code
Extension: CMake
cmake
CMake
twxs.cmake
twxs
1,793,673
Repository
License
v0.0.17
CMake langage support for Visual Studio Code
Disable
Uninstall
This extension is enabled globally.
Details
Feature Contributions
CMake For VisualStudio Code
chat on gitter
This extension provides support for CMake in Visual Studio Code.
CMakelists.txt - Visual Studio Code
File Edit View Goto Help
CMakelists.txt C:\tmp
cmake_minimum_required(VERSION 3.0)
project(Hello)
set(SOURCE main.cpp)
add_executable(${PROJECT_NAME} ${})


File Edit Selection View Go Run Terminal Help
Extension: CMake Tools - Visual Studio Code
Extension: CMake Tools
cmake
CMake Tools
ms-vscode.cmake-tools
Microsoft
1,946,151
Repository
License
v1.7.1
Extended CMake support in Visual Studio Code
Disable
Uninstall
This extension is enabled globally.
This extension is recommended based on the files you recently opened.
Details
Feature Contributions
Changelog
Extension Pack
CMake Tools
CMake Tools provides the native developer a full-featured, convenient, and powerful workflow for CMake-based projects in Visual Studio Code.
Important doc links
CMake Tools quick start
Configure and build a project with CMake Presets
Configure a project with kits and variants
Build a project with kits and variants
Debug a project
Configure CMake Tools settings
How to
FAQ
Read the online documentation
Contribute
Issues? Questions? Feature requests?
PLEASE, if you experience any problems, have any questions, or have an idea for a new feature, create an issue on the GitHub page!

xxvii. 左側のメニューから「explorer」を押し、「Open Folder」を押します。配布されたHongoフォルダーを選択して開きます。

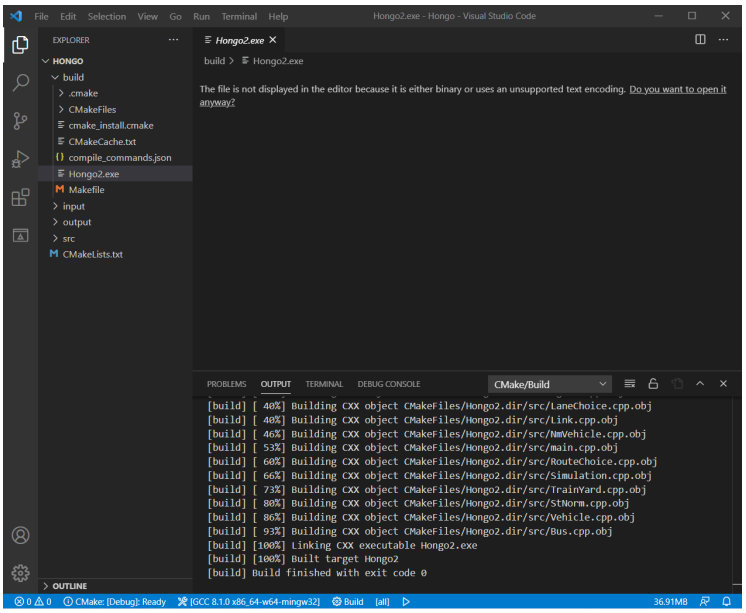


xxviii. 「F1」キーを押してコマンドパレットを開きます。「cmake」などと入力すると、「CMake: Configure」という行が出てくるので、それを押します。「build」というフォルダが作成されて、中にCMakeのキャッシュファイルなどができると思います。同時に下の「output」の欄にCMakeから出力が出てきます。



xxix. もう一度コマンドパレットを開きます。「cmake」などと入れて、「CMake: build」の行をクリックします。下の写真のような出力が出

て、「build」フォルダに「Hongo2.exe」が作成されたら成功です。



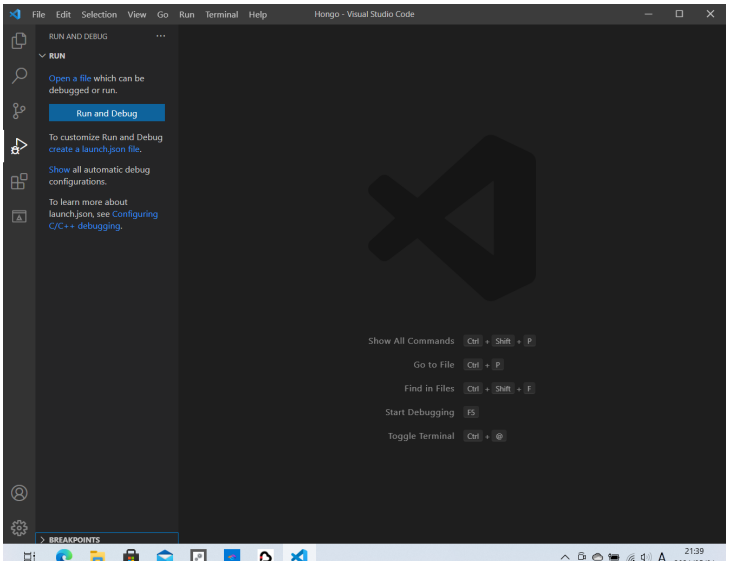
Hongo2.exeが実行ファイルになります。このままでも実行できますが、今回はVisula Studio Codeのデバッグモードで実行します。そのための設定を行います。

xxx. コマンドパレットを開きます。「C/C++」などと入れると、「C/C++: Edit Configurations(UI)」が出てくると思うので、それを選択してください。下のような画面が出てきます。コンパイラパスを先ほどインストールしたGCCのパスにしてください。（多分最初から入っているものだと思います。）パスの確認方法はコマンドプロンプトで「where gcc」→Enterで出てくるものです。

xxxi. 同じタブの「C++標準」を変更します。「gnu++17」にしてください。

xxxii. 終わったら開いていたタブを閉じます。これにより、「c_cpp_properties.json」が作成されます。

xxxiii. 左側のメニューから「Run and Debug」を押します。「Run and Debug」と書かれた青いボタンが出てくるので、それを押してください。



xxxiv. コマンドパレットが出てくるので、「C++ (Windows)」を押します。

xxxv. そうすると、勝手にlaunch.jsonが開かれるので、以下の2行を変更してください。

- 「"program": "プログラム名を入力してください (例: ${workspaceFolder}/a.exe)",」→「"program": "${workspaceFolder}/build/Hongo2.exe",」
- 「"cwd": "${workspaceFolder}",」→「"cwd": "${workspaceFolder}/build",」

終わったら上書き保存します。

以上でプログラムを実行する環境が整いました。

参考

何らかの原因（パソコンの容量が足りない場合など）で上記の環境を構築できない場合は、JetBrainsのCLionという統合開発環境も利用できます。
ここに説明されている教育用のライセンスで無料で利用することができます。
申し込み方法の詳細はここやここなどで確認できます。
インストールとアプリケーションの使い方などは、ここなどが参考になると思います。

2. パスの設定

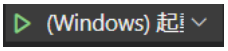
インプットとアウトプットを入れるためのフォルダーを指定します。main.cppの31行目と、DataStore.cppの38行目を、それぞれ自分のパソコンのアウトプットとインプットのパスに変更します。
※パスの最後に「/」をつけるようにお願いします。

3. 計算

a. Mac

左上の▷ボタンを押します。下のコンソールに通常出力が出てくるはずです。アウトプットフォルダのパスが出てすぐに停止してしまう場合は、2.のパスの設定があっているかもう一度確認してください。

b. Windows

左側のメニューから「Run and Debug」を押します。左上の「Start Debugging」（ 

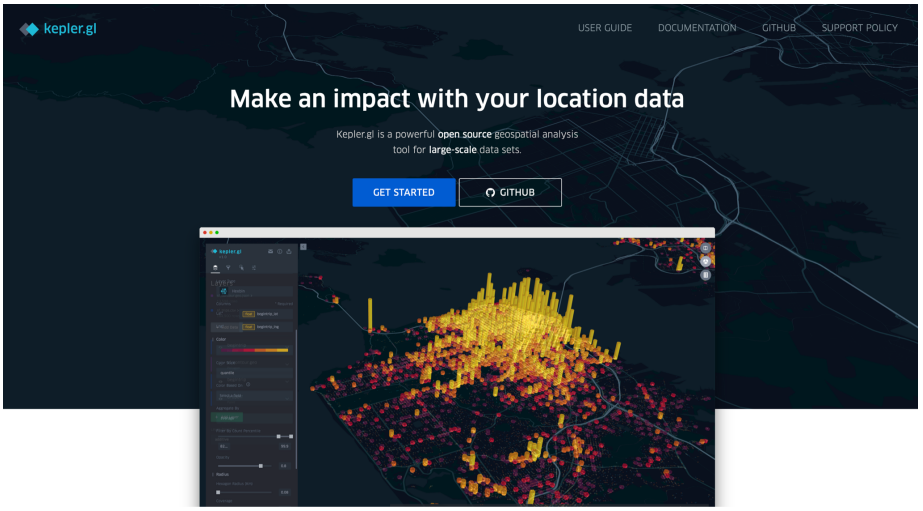
）を押すと、新しいウィンドウが開いて通常出力が表示されていくはずです。アウトプットフォルダのパスが出てすぐに停止してしまう場合は、2.のパスの設定があっているかもう一度確認してください。

4. 可視化

無事計算が終了すると、2.で設定したアウトプットフォルダに計算結果が出力されます。
今回はKepler.glを用いて結果を可視化します。Kepler.glはUBERが提供している空間情報の可視化サービスです。プログラミングフリーで高度で多彩な可視化が行えます。

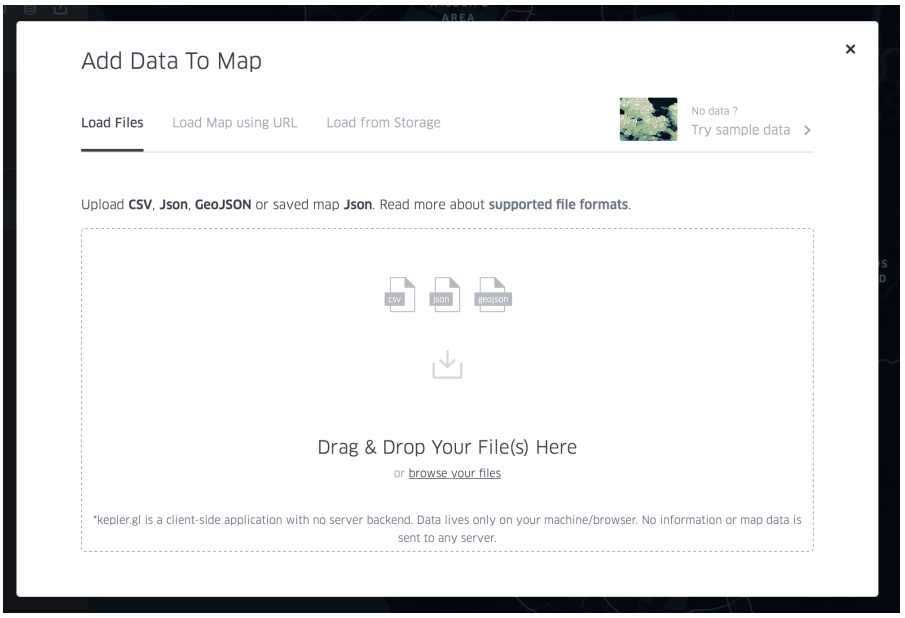
以下にその使い方を記します。

1. Kepler.glのサイトに飛びます。下のような画面が出てきます。

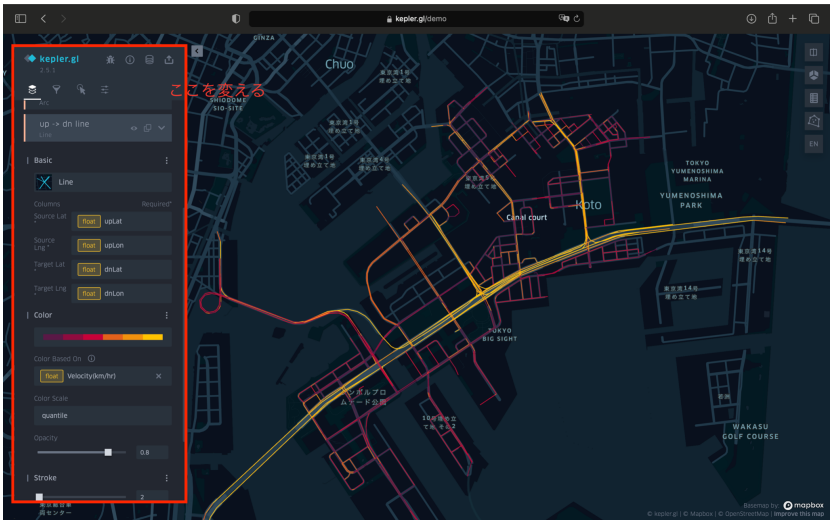


2. 「GET STARTED」を押します。あとは、ここに可視化するファイルをドラッグ＆ドロップするだけです。ファイルの形式は、csv、json、geojsonがサポートされています。csvはカラム名に「Lon」や「Lat」をつけておくと、勝手

に緯度経度を読み取ってくれます。



3. ファイルをドラッグ &ドロップすると、地図上にファイルの中身が描画されるので、左側のメニューから色や線の太さなどを好みに変えていきます。



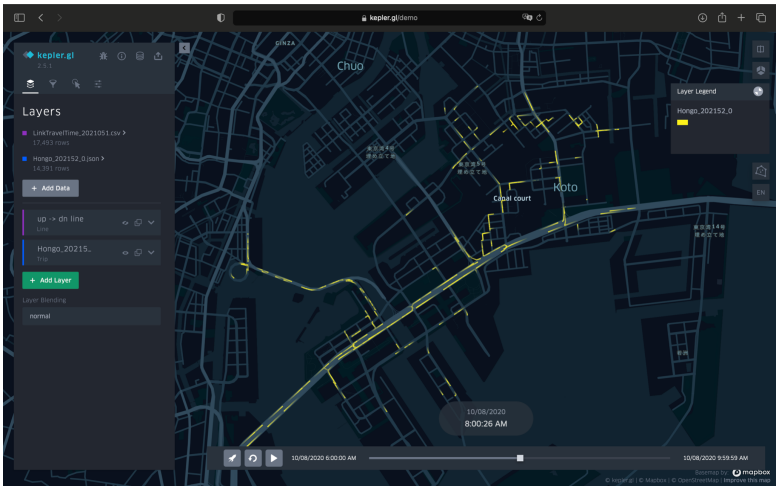
6.課題の説明

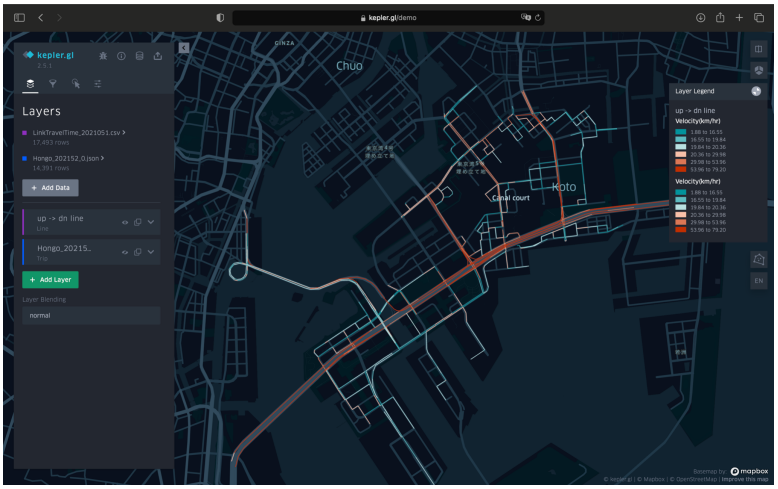
課題としては、以上の説明を参考にして、プログラムの実行とその結果の可視化の流れを一通り体験してもらいたいと思います。可視化してもらうのはアウトプットフォルダに出力される、「Hongo_[日付]_0.json」と「LinkTravelTime_[日付].csv」です。
この2つのファイルは、

- Hongo_[日付]_0.json:
  シミュレーションした自動車の軌跡。1秒ごとの自動車の位置が出力される。
- LinkTravelTime_[日付].csv:
  シミュレーションで車両が通ったリンクの旅行時間。座標が(olon, olat)の点から(dlon, dlat)の点までの線分がリンクの位置となる。1時間間隔で出力される。

なので、前者はKepler.glのTrip layerとして、後者はLine layerとして可視化してください（Kepler.glのレイヤーの説明は[こちら]）。

提出物としては、2つのレイヤーについて別々に、8:00頃の様子の写真を撮ったものとします（下の写真のようなものです）。Trip layerの色や、線の太さ、背景の地図などは好みで良いですが、「LinkTravelTime_[日付].csv」については色を「Velocity(km/h)」に応じて変えるようにしてください。





※ヒント：時間の情報が入っているLine layerの場合は、左側のメニューの「Filters」から「Add Filter」を押すと、特定の時間だけを表示させることができます。