プログラミングC言語Ⅱ期末レポート

学籍番号　1116191025

氏名　　　喜多尚之

※アプリの使い方・概要・目的に関してはREADME.txtを参照してください

# アプリのフローチャート

ダイアグラム

自動的に生成された説明

図1. アプリのフローチャート

迷路作成のアルゴリズムについては後述。

# ソースファイルごとの説明

## “PDCursesSample.cpp”

　メイン関数を実行するファイル(Cursesがサンプルファイルを複製することでしか起動できないので、名前が“PDCursesSample.cpp”のまま)。

## “functions.cpp”

　メイン関数で使用する関数の定義を記述したファイル。

## “Header.h”

　構造体Mazeの定義、”functions.cpp”で定義する関数のプロトタイプ宣言を記述したファイル。

# アプリのこだわり、工夫点、コーディングではまった点

## アプリのこだわり

　関数や構造体の定義にわかりやすいようにコメントアウトを記述してわかりやすいようにしています。

## 工夫点

・構造体で2次元配列を使用する際、構造体の定義内で要素数を指定せずに宣言だけしておくことが不可能だったので、十分な大きさの2次元配列を定義しておき、初期化時にその配列の1部分を使用するようにしました。

・再帰関数を用いて迷路作成のアルゴリズムを実装しました。以下に説明します。

i)アルゴリズム

・穴掘り法とは、迷路を作成するためのアルゴリズムです。例を挙げて説明します。

・まず、迷路のすべてのマスを壁で埋めます。(■…壁)

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

・次に、ランダムに座標が奇数のマスを1つ掘り、道にします。(○…今掘ったマス)

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■○■

　■■■■■■

・堀ったマスについて、隣り合うマス(上下左右)から1つランダムに決め、掘ることができれば掘り、できなければ他のマスについて同様の作業をします。

　ここで、掘ることができるマスとは、以下の条件を満たすマスを指します。

１．迷路の縁じゃない

２．そのマスを掘ることで別の道とつながらない

　例えば、先ほどの状況でランダムに決めた結果、右のマスだったとすると、迷路の縁なので掘ることができません。

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■　■←迷路の縁なので無理

　■■■■■■

　よって、他のマスから同様にランダムに決めます。左のマスを試してみると、条件を満たしているので掘ることができます。

　■■■■■■

　■■■■■■

　■■■■■■

　■■■■■■

　■■■○　■

　■■■■■■

・前の処理で掘ったマスについて、同様に処理を行います。つまり、隣り合うマスからランダムに選び、掘ることができたら掘ります。

　■■■■■■

　■■■■■■

　■■■■■■

　■■■○■■

　■■■　　■

　■■■■■■

・すべてのマスについて掘ることができなくなるまで続けます。

　■■■■■■　　　■■■■■■　　　■■■■■■

　■■■■■■　　　■■■■■■　　　■■■■■■

　■■■■■■　→　■■■■■■　→　■■■■■■　→もう掘れない

　■■○　■■　　　■○　　■■　　　■　　　■■

　■■■　　■　　　■■■　　■　　　■○■　　■

　■■■■■■　　　■■■■■■　　　■■■■■■

・掘れなくなったら先ほど掘ったひとつ前のマスに戻り、他の方向で掘れないか試します。掘れたらそのまま進みます。(★…今いるマス)

　■■■■■■　　　■■■■■■　　　■■■■■■　　　■■■■■■

　■■■■■■　　　■■■■■■　　　■○■■■■　　　■　○■■■

　■■■■■■　→　■○■■■■　→　■　■■■■　→　■　■■■■

　■★　　■■　　　■　　　■■　　　■　　　■■　　　■　　　■■

　■　■　　■　　　■　■　　■　　　■　■　　■　　　■　■　　■

　■■■■■■　　　■■■■■■　　　■■■■■■　　　■■■■■■

　上に掘れる

　　　　　　　　　　■■■■■■　　　■■■■■■　　　■■■■■■

　　　　　　　　　　■　　○■■　　　■　　　○■　　　■　　　　■

　　　　　　　　→　■　■■■■　→　■　■■■■　→　■　■■○■

　　　　　　　　　　■　　　■■　　　■　　　■■　　　■　　　■■

　　　　　　　　　　■　■　　■　　　■　■　　■　　　■　■　　■

　　　　　　　　　　■■■■■■　　　■■■■■■　　　■■■■■■

・前の処理(ひとつ前のマスに戻って他の方向を確認)を繰り返していき、最初の地点の確認が完了したら終了です。

　■■■■■■　　　■■■■■■　　　　　　　■■■■■■　　　■■■■■■

　■　　　★■　　　■　　★　■　　　　　　　■　　　　■　　　■　　　　■

　■　■■　■　→　■　■■　■　→　…　→　■　■■　■　→　■　■■　■

　■　　　■■　　　■　　　■■　　　　　　　■　　　■■　　　■　　　■■

　■　■　　■　　　■　■　　■　　　　　　　■　■　★■　　　■　■　　■

　■■■■■■　　　■■■■■■　　　　　　　■■■■■■　　　■■■■■■

　もう掘れない　　　もう掘れない　　　　　　もう掘れないので…　　　完成！

ii)実装

dig関数とdig\_maze関数を用いて実装しました。dig関数は掘ることができれば掘る関数、dig\_mazeはdig関数を用いた再帰的関数です。

dig\_maze関数の構造を簡単に説明します。

1. 方向(上下左右)を指し示す配列をシャッフルします。

2. 1.でシャッフルした配列に順番にアクセスし、その方向にある隣り合ったマスに対してdig関数を適用します。

3. 掘ることができたら、そのマスに関してdig\_maze関数を適用します。できなかったら何もしません。

例を挙げて説明します。

　■■■■■■

　■○■■■■

　■■■■■■

空いているマスに対してdig\_maze関数を用います。すると

1-1. 配列のシャッフル({UP, RIGHT, LEFT, DOWN}になったとします)

1-2. UP, RIGHT→掘れない　LEFT→掘れる　掘ってdig\_maze関数を呼び出す

　　　　　　■■■■■■

　　　　　　■　○■■■

　　　　　　■■■■■■

2-1. 配列のシャッフル({DOWN, LEFT, UP, RIGHT})

2-2. DOWN→掘れない　LEFT→掘れる　掘ってdig\_maze関数を呼び出す

　　　　　　　　　　■■■■■■

　　　　　　　　　　■　　○■■

　　　　　　　　　　■■■■■■

3-1.配列のシャッフル({LEFT, UP, DOWN, RIGHT})

3-2. LEFT→掘れる　掘ってdig\_maze関数を呼び出す

　　　　　　　　　　 ■■■■■■

　 　　 　 　 　　　　　■　　　○■

　　　 　 　 　 　　　　■■■■■■

4-1. 配列のシャッフル({LEFT, RIGHT, UP, DOWN})

4-2. LEFT, RIGHT, UP, DOWN→掘れない

4-3. dig\_maze関数を終了します

　　　　　　　　　　 ■■■■■■

　 　　 　 　 　　　　　■　　★　■

　　　 　 　 　 　　　　■■■■■■

3-3. UP, DOWN, RIGHT→掘れない

3-4. dig\_maze関数を終了します

　　　　　　　　　　■■■■■■

　　　　　　　　　　■　★　　■

　　　　　　　　　　■■■■■■

2-3. UP, RIGHT→掘れない

2-4. dig\_maze関数を終了します

　　　　　　■■■■■■

　　　　　　■★　　　■

　　　　　　■■■■■■

1-3. DOWN→掘れない

1-4. dig\_maze関数を終了します

以上のような挙動になります。アルゴリズムに忠実に動いていると思います。

※その他に関してはGithubのissueを確認してください

## コーディングではまった点

・PDCursesで、2つの全角文字を表示する際、y座標を十分に開けなかったため重なって文字化けしてしまった点

・print\_maze関数で、refreshのタイミングが不正であったため文字化けした点

・カレントディレクトリがどこにあるのか不明で入出力ファイルの設定ができなかった点

※その他に関してはGithubのissueを確認してください

# 更なる改良点

・dig\_mazeの軽量化

　現在のdig\_maze関数は画面にちらつきが存在するので、軽量化してちらつきを軽減したい。

・ファイルの細分化

　現在、関数はすべて”functions.cpp”に記述されているが、使用するシーンや用途に合わせて分けたい。

・最短経路の表示

　プレイヤーの移動を可変長リストに保存しておき、クリア時にその移動履歴から余剰な移動を削除することによって最短経路を求めることができるので、実装したい。