実験２（ソフトウェア）資料

目次

1. 概要

1.1. この実験の目標

計算機科学実験１では、Javaプログラミング言語の基礎を学ぶとともに、探索やソートなどの基本的なアルゴリズムの実装を行った。この実験（実験２SW）では、実験１で演習したプログラミングおよびアルゴリズムに関する知識を前提に、具体的なタスクに活かすにはどうするか、より大規模かつ実際的なプログラミング演習を通じて学ぶことを目的とする。具体的には、ソースコードのバージョン管理システムであるgitの使用方法、ゲームエージェントのプログラミングを通じて探索・ソートなどのアルゴリズムを適用する手法について学び、考える。

1.2. 注意

資料に関する注意

実際に課題を始める前に，必ずこの資料の当該 部分までを通して読んでおくこと．この予習は実験の時間外に行なうことを想定している．実験時間になって初めて資料を読み始めていると間違いなく時間が足りなくなるので注意すること．実験時間は「実際にプログラムを 書く時間」だと考えること．

Javaの学習について重要な注意

本実験では，Java の基礎に関しては細 かい説明は行なわないので，実験１の資料および予め指定教科書などで自習しておくこと．

成績の付け方

報告書（レポート），総合デモ，出席に基いて行う．必修の課題は4つあり，この全てを完了することが必要である．

報告書：以下の2つの報告書を提出する．
報告書1：課題1, 2
報告書2：課題3, 4

それぞれの課題の内容をレポートにまとめ，作成したプログラムとともに提出する。以降の「レポートの書き方」を参考にすること。

総合デモ：実験で作成したプログラムの動作を実際に確認する．付録Aを参照せよ．
出席：本実験は出席が義務付けられている．正当な理由なく 一定時間以上欠席・遅刻すると単位は与えられない．

発展課題については加点対象とする．本資料に記載されている内容に関わらず，自由な人工知能アルゴリズムを適用して構わない． ただし、その中身の解説、および参考にした資料やURLなどはきちんとリファレンスすること（当然ながら他所のコピーは厳禁）

報告書２は、課題が全部完了しなかった場合でもどの程度進められたかを記述し、一旦締切日までに提出すること。 完成度によっては再提出を求めることがある。ただし、締切日以降であっても発展課題の完成度を挙げられた場合には、後期の実験２HW の終了日までであれば再提出を認める。この場合、発展課題の内容によって加点を加える。

1.3. 報告書の書き方について

提出方法・内容

PandAで提出すること。提出物は以下のとおりである。
・報告書本文（テキストあるいはPDF）
・プログラムソースコード（.javaファイルまたは複数ファイルの場合は書庫(zip,tar.gz等）ファイル。

提出物の形式

報告書本文および該当するプログラムのソースコードを提出すること。
報告書の本文は、テキストあるいはPDFファイルとする。内容は、以下のようにすること。

- 報告書タイトル（例、計算機科学実験２ソフトウェア報告書１、計算機科学実験２ソフトウェア報告書２、計算機科学実験２ソフトウェア報告書３）
- 氏名・入学年度・学籍番号
- 以下は、「実施内容、実行（実装）結果と解説、結論と考察」を記述すること。課題が複数あるので、その課題数に応じて「実施内容、実行結果、結論と考察」を書くこと。
- 図や表などを入れてわかりやすくするよう工夫すること。
- 考察は「うまく行った（行かなかった）」の単純な結果のみでなく、どうしてうまく行かなかったのか、予想通り動いたのか、どういったことをすると良くなるか、などの考えを書くこと。図は各レポートで３個以上、文字数は１課題に対し４００文字以上は書くこと。

報告書内容の確認

提出された報告書はTAがチェックする。TAから再提出されるよう指示される場合があるので、PandAを定期的にチェックすること。指摘された内容を修正し、再提出すること。

課題提出に関する注意事項

万が一時間が足りなくて課題の提出期限までに全部出来なかった場合，全部できていないことを明記し，できた所までの内容 で提出すること(当然，評価は下がる)。特に報告書２は、課題が全部完了しなかった場合でもどの程度進められたかを記述し、一旦締切日までに提出すること。 完成度によっては再提出を求めることがある。ただし、締切日以降であっても発展課題の完成度を挙げられた場合には、後期の実験２HW の終了日までであれば再提出を認める。この場合、発展課題の内容によって加点を加える。

2. バージョン管理システムGITの使用方法とマリオAIの設定

2.1. バージョン管理システムとは

バージョン管理システムとは、コンピュータ上で作成、編集されるファイルの変更履歴を管理するためのシステム。特にソフトウェア開発においてソースコードの管理に用いられる。

バージョン管理システムの最も基本的な機能は、ファイルの作成日時、変更日時、変更点などの履歴を保管することである。これにより、何度も変更を加えた ファイルであっても、過去の状態や変更内容を確認したり、変更前

の状態を復元することが容易になる。更に、多くのバージョン管理システムでは、複数の人間 がファイルの編集に関わる状況を想定している。商業的なソフトウェア開発やオープンソースプ ロジェクトなどでは、複数の人間が複数のファイルを各々編集するため、それぞれのファイルの最新の状態が分からなくなったり、同一ファイルに対する変更が 競合するなどの問題が生じやすいが、バージョン管理システムは、このような問題を解決する仕組みを提供する。ただし、バージョン管理システムを個人のファ イル管理に使用することも可能であるし、ソフトウェアのソースコードだけでなく、設定ファイルや原稿の管理などにも使うことも可能である。

バージョン管理システムでは、ファイルの各バージョンをデータベースに保持しており、このデータベースを一般にリポジトリと呼ぶ。バージョン管理システムの基本的な利用方法は以下の流れになる。

1. ファイルをリポジトリに登録する。
2. ファイルをリポジトリからローカル環境に取り出す（チェックアウト）
3. ローカル環境で、ファイルに対し変更を行う。
4. 変更したファイルをリポジトリに書き戻す（チェックイン）
5. ファイルがチェックインされると、システムによって「いつ」「誰が」「どんな変更を行った」等が記録され、後から参照できる。また必要に応じて古い版を取り出すことも出来る。

(Wikipedia:バージョン管理システム)。

2.2. GITとは

git（ギット）は、分散型バージョン管理システムである。Linuxカーネルの開発者リーナス・トーバルズによって開発され、多くのプロジェクトで採用されている。Linuxカーネルのような巨大プロジェクトにも対応できるように、動作速度に重点が置かれている。gitでは、各ユーザのワーキングディレクトリに、全履歴を含んだリポジトリの完全な複製が作られる。したがって、ネットワークにアクセスできないなどの理由で中心リポジトリにアクセスできない環境でも、履歴の調査や変更の記録といったほとんどの作業を行うことができる。

Linux上でのGITの代表的なコマンドとして以下のようなものがある。

cd <dir>; git init	新たなgitレポジトリを作成する
git clone <URL>	gitレポジトリのクローンを作成する
git status	変更が行われたファイルを表示する
git diff	変更部分をdiff形式で表示する
git add	コミットするファイルを指定する
git commit	変更点をコミットする
git log	コミットログを表示する
git reset	直接のコミットを取り消す
git branch <name>	新たなブランチを作成する
git branch	ブランチを表示する
git checkout <name>	nameであらわされたブランチで作業する

git push <URL> <source name>:<dest name>
ブランチをレポジトリに送信する

2.3. GITレポジトリからのマリオAIのダウンロード(git clone)

計算機科学コースでは、計算機科学実験用にgitサーバを立てており、ここから本実験用に設定された最新のコードをダウンロードできる。詳しくは別添付のスライドを参照すること。

3. 「マリオAI」によるJAVAプログラミング演習

3.1. マリオAI概要

マリオAIは、ゲームAIの技術を競う目的で開発されたベンチマークソフトウェアである。このベンチマークソフトウェアを利用した競技会が、2009年から2012年まで毎年開催され、ゲームAI技術の発展に貢献した。本演習では、http://130.54.13.121/~le2soft/2017marioAI.tar.gz （実験室PCからのみアクセス可）で公開されているプロジェクトを用いる。

3.2. マリオの操作方法

そのまま、マリオをプレイできる。操作方法は下記：

キー	説明
A	ファイア＆ダッシュ
S	ジャンプ
←	左へ移動
→	右へ移動
↓	下へ移動
W	終了

3.3. 表示の設定と画面表示内容

表示の設定

キー	説明
8	ゲーム速度を最大にする／元に戻す
スペース	一時停止／再開
C	マリオが画面中央になるようカメラを移動
G	グリッド表示／非表示
Z	画面拡大／元に戻す

画面表示内容

画面表示	内容
DIFFICULTY	ステージ難易度
SEED	ステージ生成のシード
TYPE	ステージ種類
LENGTH	マリオの横方向位置
HEIGHT	マリオの縦方向位置（数字が小さいほど上にいる）
コインアイコン	獲得コイン数
キノコアイコン	獲得キノコ数
フラワーアイコン	獲得フラワー数
Agent	エージェント名
PRESSED KEYS	入力キー
ALL KILLS	倒した敵の数
by Fire	ファイアで倒した敵の数
by Shell	甲羅で倒した敵の数
by Stomp	踏みつけで倒した敵の数
TIME	残り時間

FPS	ゲーム速度

4. エージェントプログラミングの基礎

4.1. エージェントとステージパラメータの設定

4.1.1. エージェントの設定

人間ではなくエージェントによってマリオを動作させる方法を解説する。ch.idsia.scenarios.Main.javaに以下の変更を加えると、実装済みの”FowardJumpingAgent”使ってゲームを実行できる。

Step 1.

```
import ch.idsia.tools.MarioAIOptions;
```

の次に

```
import ch.idsia.agents.Agent;
import ch.idsia.agents.controllers.ForwardJumpingAgent;
```

と記述する。

Step 2.

```
final MarioAIOptions marioAIOptions = new
MarioAIOptions(args);
```

の次に

```
final Agent agent = new ForwardJumpingAgent();
marioAIOptions.setAgent(agent);
```

と記述する。

以上の変更を加えた上でch.idsia.scenarios.Main.javaを実行すると、エージェントによってマリオが自動的に動作する（画面に”Agent: ForwardJumpingAgent”と表示される）。なお、変更を加えた後のコードがch.idsia.scenarios.Main2.javaである。参考にされたい。

4.1.2. ステージパラメータの設定

ch.idsia.scenarios.Main.javaで様々なステージパラメータを設定することで、ステージの種類を変えることができる。

ステージのランダム生成

seedの値を変更することで、ステージをランダムに変更できる。

```
int seed = 99;
marioAIOptions.setLevelRandSeed(seed);
```

難易度

```
marioAIOptions.setAgent(agent);
```

の次の行に以下を記述すると、難易度の高いステージとなる。

```
int d = 100;
marioAIOptions.setLevelDifficulty(d);
```

dの値を変更することで、難易度を調整することができる。ただし、dは0以上の整数である。

敵の有無

```
marioAIOptions.setAgent(agent);
```

の次の行に以下を記述すると、キラーとパックンフラワー以外の敵が出現しないステージになる。

```
marioAIOptions.setEnemies("off");
```

また、敵コードを使って、出現する敵の種類を細かく制御できる。

例1：クリボーだけが出現するステージ

```
marioAIOptions.setEnemies("g");
```

ここで、”g”がクリボーを表す敵コードである。

例2：クリボーと緑ノコノコだけが出現するステージ

```
marioAIOptions.setEnemies("ggk");
```

ここで、”g”がクリボー、”gk”が緑ノコノコを表す敵コードである。
その他の敵コードを以下に示す。

敵	敵コード
クリボー	g
パタクリボー	gw
緑ノコノコ	gk
緑パタパタ	gkw
赤ノコノコ	rk
赤パタパタ	rkw
トゲゾー	s
羽トゲゾー	sw

その他
その他のステージパラメータの設定方法を以下に示す。サンプルコードがch.idsia.scenarios.Main3.javaにある。参考にされたい。

メソッド	説明
`marioAIOptions.setDeadEndsCount(boolean var);`	行き止まりの有無
`marioAIOptions.setCannonsCount(boolean var);`	キラー砲台の有無
`marioAIOptions.setHillStraightCount(boolean var);`	丘の有無
`marioAIOptions.setTubesCount(boolean var);`	土管の有無
`marioAIOptions.setGapsCount(boolean var);`	落とし穴の有無
`marioAIOptions.setHiddenBlocksCount(boolean var);`	隠しブロックの有無
`marioAIOptions.setBlocksCount(boolean var);`	ブロックの有無
`marioAIOptions.setCoinsCount(boolean var);`	コインの有無
`marioAIOptions.setFlatLevel(boolean var);`	地面を平面にするか否か

4.2. サンプルエージェントプログラムの説明

ch.idsia.agents.controllers.ForwardJumpingAgent.javaを例に、エージェント

プログラムの方法を解説する。

reset()メソッド
reset()メソッドは、ゲームの開始時に呼ばれるメソッドである。
ForwardJumpingAgentでは、まず、6次元のboolean配列を作成している。

```
action = new boolean[Environment.numberOfKeys];
```

action配列の各要素は、以下のボタンに対応している。trueに設定されると、そのキーを押していることになる。初期状態では、全ての要素はfalseに設定されている。

変数	対応ボタン
action[Mario.KEY_LEFT]	左移動
action[Mario.KEY_RIGHT]	右移動
action[Mario.KEY_DOWN]	しゃがむ
action[Mario.KEY_JUMP]	ジャンプ
action[Mario.KEY_SPEED]	ダッシュ＆ファイア
action[Mario.KEY_UP]	上移動

ForwardJumpingAgentでは、以下のように設定されている。これにより、ゲーム開始時には「右移動」と「ダッシュ＆ファイア」ボタンが押され、マリオは右方向へダッシュする。

```
action[Mario.KEY_RIGHT] = true;
action[Mario.KEY_SPEED] = true;
```

getAction()メソッド
getAction()メソッドは、ターン毎に呼ばれるメソッドである。そのターンでのマリオの行動を決定し、出力する。ForwardJumpingAgentでは、以下のように設定されている。

```
action[Mario.KEY_SPEED] = isMarioAbleToJump || !isMarioOnGround;
action[Mario.KEY_JUMP] = isMarioAbleToJump || !isMarioOnGround;
```

isMarioAbleToJump は、マリオがジャンプ可能なときtrue、それ以外のときfalseとなる変数である。また、isMarioOnGround は、マリオが地上にいるtrue、それ以外のときfalseとなる変数である。”isMarioAbleToJump || !isMarioOnGround” は「マリオがジャンプ可能なとき、または、マリオが空

中にいる」場合はtrueとなり、「マリオがジャンプ不可能で、かつ、地上にいる」場合はfalseとなる。

つまりForwardJumpingAgentは、マリオが地上にいてジャンプできないときは「ジャンプ」と「ダッシュ＆ファイア」ボタンを離す。それ以外の場合は「ジャンプ」と「ダッシュ＆ファイア」ボタンを押す。マリオの着地後に再びジャンプをさせるため、一度ジャンプボタンを離している。

なお、以下のようにすると、着地後にマリオは再びジャンプすることができない。

```
action[Mario.KEY_SPEED] = action[Mario.KEY_JUMP] =true;
```

4.3. 環境情報へのアクセス

getAction()の中で以下のメソッドを用いることで、マリオの周辺位置のマップ、敵の情報を取得することができる。

getReceptiveFieldCellValue()メソッド

引数r, cで指定された場所のマップの値を返すメソッド。rは行番号, cは列番号である。

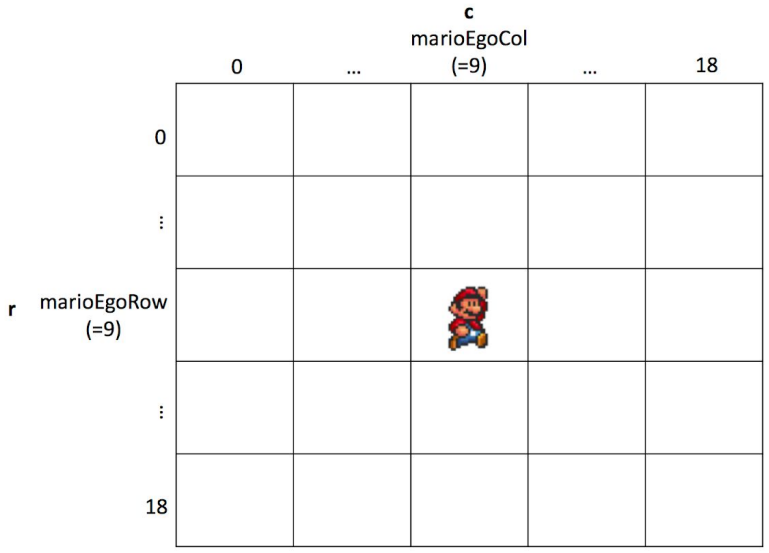
```
getReceptiveFieldCellValue(int r, int c)
```

getEnemiesCellValue()メソッド

引数r, cで指定された場所の敵の値を返すメソッド。

```
getEnemiesCellValue(int r, int c)
```

なお、r, cとして指定できるのは0, 1, …, 18である。それ以外の値を指定した場合、”0”が返ってくる。



getReceptiveFieldCellValue()が返す値の意味を次の表に示す。

値	定数	説明
-24	GeneralizerLevelScene.BRICK	ブロック
2	GeneralizerLevelScene.COIN_ANIM	コイン
-60	GeneralizerLevelScene.BORDER_CANNOT_PASS_THROUGH	通過できない障害物
-62	GeneralizerLevelScene.BORDER_HILL	通過可能な障害物（下から通過し上に乗れる）
-85	GeneralizerLevelScene.FLOWER_POT_OR_CANNON	土管、キラー砲台
61	GeneralizerLevelScene.LADDER	はしご
5	GeneralizerLevelScene.PRINCESS	ピーチ姫
0		障害物なし

getEnemiesCellValue()が返す値の意味を次の表に示す。

値	定数	説明
0	Sprite.KIND_NONE	敵なし
-31	Sprite.KIND_MARIO	マリオ
80	Sprite.KIND_GOOMBA	クリボー

95	Sprite.KIND_BOOMBA_WINGED	羽クリボー
82	Sprite.KIND_RED_KOOPA	赤ノコノコ
97	Sprite.KIND_RED_KOOPA_WINGED	赤パタパタ
81	Sprite.KIND_GREEN_KOOPA	緑ノコノコ
96	Sprite.KIND_GREEN_KOOPA_WINGED	緑パタパタ
84	Sprite.KIND_BULLET_BILL	キラー
93	Sprite.KIND_SPIKY	トゲゾー
99	Sprite.KIND_SPIKY_WINGED	羽トゲゾー
91	Sprite.KIND_ENEMY_FLOWER	パックンフラワー
13	Sprite.KIND_SHELL	甲羅
2	Sprite.KIND_MUSHROOM	キノコ
3	Sprite.KIND_FIRE_FLOWER	フラワー
25	Sprite.KIND_FIREBALL	ファイアボール

4.4. その他の情報へのアクセス

以下の変数にマリオの様々な情報が格納されている

- marioStatus
  - 0: DEAD, 1: WIN, 2: RUNNING
- marioMode
  - 0: SMALL, 1: LARGE, 2: FIRE
- isMarioOnGround
- isMarioAbleToJump
- isMarioAbleToShoot
- isMarioCarrying
- getKillsTotal
- getKillsByFire
- getKillsByStomp
- getKillsByShell
- distancePassedCells
- distancePassedPhys
- flowersDevoured
- mushroomsDevoured
- coinsGained
- timeLeft
- timeSpent
- hiddenBlocksFound

4.5. 課題１

ステージパラメータを様々に変更し、動作を確認・理解する。また、以下の既に実装されているエージェントの１つを選び、ソースコードから動作を説明せよ。ソースコードは、ch.idsia.agents.controllers.[エージェント名].javaにある。

- ForwardAgent
- RandomAgent
- ScaredShootyAgent

5. 敵のないシーンでのエージェントプログラミング

5.1. 自分のエージェントを作成・動作させる

エージェントのテンプレート”OwnAgent”が
ch.idsia.agents.controllers.OwnAgent.javaに実装されている。
4.4.1節と同様に、ch.idsia.scenarios.Main2.javaで”ForwardJumpingAgent”となっている箇所をOwnAgent に書き換えることで、自分のエージェントを読み込むことができる。サンプルコードがch.idsia.scenarios.MainOwn.javaにあるので参照されたい。書き換えた後のMain2.java（あるいはMainOwn.java）を実行すると、ゲームが起動するがマリオは何もしない。

5.2. reset()メソッドの変更

OwnAgent.javaのreset()メソッドを以下のように書き換える

```
public void reset()
{
    action = new boolean[Environment.numberOfKeys];
    action[Mario.KEY_RIGHT] = true;
}
```

Main2.java（あるいはMainOwn.java）を実行すると、「右移動キー押しっぱなしのマリオ」となる

5.3. getAction()メソッドの変更

OwnAgent.javaのgetAction()メソッドを以下のように書き換える

```
public boolean[] getAction()
{
    Random R = new Random();
    action[Mario.KEY_DOWN] = R.nextBoolean();

    return action;
}
```

また、Randomをインポートする。

```
import java.util.Random;
```

Main2.java（あるいはMainOwn.java）を実行すると、「右移動キー押しっぱなし＆ランダムにしゃがむ」マリオとなる。サンプルコード：ch.idsia.agents.controllers.OwnAgent2.java

5.4. 障害物をよけながらクリアするエージェントのプログラミング

getAction()メソッドを以下のようににすることで、目の前に障害物がある場合にジャンプして避けるエージェントとなる。

```
public boolean[] getAction()
{
	if(isObstacle(marioEgoRow, marioEgoCol + 1) ||
			getEnemiesCellValue(marioEgoRow, marioEgoCol +
2) != Sprite.KIND_NONE
			|| getEnemiesCellValue(marioEgoRow, marioEgoCol
+ 1) != Sprite.KIND_NONE){
		action[Mario.KEY_JUMP] = isMarioAbleToJump ||
!isMarioOnGround;
	}
    return action;
}
```

下記の条件式は、マリオの目の前に何かしら障害物（ブロック、通過できない障害物、土管）がある場合、trueとなる。

```
isObstacle(marioEgoRow, marioEgoCol + 1)
```

isObstacle関数は以下のように定義されている：

```
public boolean isObstacle(int r, int c){
	return getReceptiveFieldCellValue(r,
c)==GeneralizerLevelScene.BRICK
			|| getReceptiveFieldCellValue(r,
c)==GeneralizerLevelScene.BORDER_CANNOT_PASS_THROUGH
			|| getReceptiveFieldCellValue(r,
c)==GeneralizerLevelScene.FLOWER_POT_OR_CANNON
			|| getReceptiveFieldCellValue(r,
c)==GeneralizerLevelScene.LADDER;
}
```

下記の条件式は、マリオの２つ前に敵がいる場合、trueとなる。

```
getEnemiesCellValue(marioEgoRow, marioEgoCol+2)!= Sprite.KIND_NONE
```

下記の条件式は、マリオの目の前に敵がいる場合、trueとなる。

```
getEnemiesCellValue(marioEgoRow, marioEgoCol+1)!= Sprite.KIND_NONE
```

以上によりif文の中身がtrueのとき、つまりマリオの目の前に障害物がいいる場合、または目の前か２つ前に敵がいるときにジャンプを行うエージェントとなる。サンプルコード：
ch.idsia.agents.controllers.IgnoreObstacleAgent.java

5.5. 課題２

上記のアルゴリズムを拡張し（穴をジャンプでよける、ブロックで進行が止められる状況を回避し）、与えられた敵のないシーン（ch.idsia.scenarios.MainTask2.java）でステージをクリアするエージェントを実装し、クリアできることを確認せよ。どのようなプログラミングを行ったかのアイデアと実装方法をレポートで示すこと。

6. 敵のあるシーンでのエージェントプログラミング

6.1. 課題3

「敵を避けつつクリアする」エージェントの実装を行い、ch.idsia.scenarios.MainTask3.javaを解けるエージェントつくれ。「敵を攻撃する」も実装できればなお良いが必要条件ではない。どのようなプログラミングを行ったかのアイデアと実装方法をレポートで示すこと。レポートは以下のように構成すること。

（１）現象の観察
ch.idsia.scenarios.MainTask3.javaに対して、課題２で作成したエージェントを動作させ、結果を観察・記述すること。
（２）原因の解析
（１）の結果となった原因を考察し、解決策を考える。
（３）実装
（２）で考案した解決策を実装すること。コードを添付し、解説を加えること。
（４）検証
（３）で実装した結果を動作させ、面がクリアできることを確認すること。

7. 発展課題

7.1. より高度な動作生成

7.1.1. ルールベース

「目の前に障害物がある場合にジャンプして避ける」などのルールを複数組み合わせることで、複雑なマリオの動作を表現することができ

る。

7.1.2. モンテカルロ法

モンテカルロ法は強化学習の一手法である。各状態 s に対する各行動 a の価値 $Q_{s,a}$ を学習し、行動選択に用いる。各エピソード（ゲーム開始から終了まで）i の終了時点で報酬 r_t を得るとする。各エピソードが終わる度に、エピソード中で現れた状態 s とそのときの行動 a について、$Q_{s,a}$ を次式で更新する：

$$Q_{s,a} \leftarrow \frac{1}{|E_s|} \sum_{i \in E_s} r_i$$

ここで E_s は、状態sが登場したエピソードの集合である。また更新は、状態 s に対する当該エピソードでの初回の行動 a のみに対して行なう。

モンテカルロ法の実装例がch.idsia.agents.LearningWithMC.java にあり、ch.idsia.idsia.scenarios.champ.LearningTrackMC.java を実行することで動作を確認できる。152～158行目のnew LearningWithMC()で目的の面を設定すること。

7.1.3. 遺伝的アルゴリズム

遺伝的アルゴリズムとは、主要な進化的アルゴリズムの1つである。
遺伝的アルゴリズムでは、解の候補を複数用意し、目的関数のスコアの高い解の候補を優先的に選択し、解の候補同士の交叉、突然変異などの操作を繰り返し新たな解の候補を生成する。これらを繰り返すことによって目的関数を最適化し、解を探索する。
アルゴリズムは以下のようになる。個体数をN、最大世代数をGとおく。

1. N個の解の候補の集合、「現世代」と「次世代」をそれぞれ生成する。
2. N個の解の候補をランダムに生成し、現世代に入れる。
3. 目的関数によって現世代のN個の解の候補のスコアを計算する。
4. 事前に設定した確率で以下の動作のいずれかをおこない、次世代の解の候補をN回生成する。
    a. 現世代に含まれる個体を2つ選択し、交叉を行う
    b. 現世代に含まれる個体を1つ選択し、突然変異を行う
    c. 現世代に含まれる個体を1つ選択し、コピーする
5. 現世代の集合に次世代の集合を代入し、世代の回数がGに満たない場合3に戻る
6. 「現世代」の中で最も適応度の高い個体を「解」として出力する

遺伝的アルゴリズムでの実装例が
ch.idsia.agents.LearningWithGA.java にあり、
ch.idsia.idsia.scenarios.champ.LearningTrack.java を実行することで

動作を確認できる。215～221行目のnew LearningWithGA()で目的の面を設定すること。

参考：

- https://ja.wikipedia.org/wiki/遺伝的アルゴリズム
- 電気通信大学 橋山研究室作成「マリオAIマニュアル」 http://www.media.is.uec.ac.jp/medialab-wp/imlab/wp-content/blogs.dir/5/files/2012/12/MarioAI_Manual_005.pdf

7.1.4. その他

A*がMarioAIで高いパフォーマンスを発揮することが知られている。

参考：

- https://ja.wikipedia.org/wiki/A*
- https://www.youtube.com/watch?v=DlkMs4ZHHr8

7.1.5. 注意

最終デモでは，課題がクリアできるかを確認する．機械学習などを使ったエージェントを設計した場合は，学習結果を保存・ロードすることで，すぐにデモできるように設計すること．

7.2. 課題４

以下の基本課題および発展課題に取り組み、動作状況及び内容をレポートを説明せよ。

基本課題（最低限、この課題が解けるようにする）

`ch.idsia.scenarios.MainTask4_1.java`を解くエージェントを作れ。最低限ルールベース、可能であればそれ以上の発展アルゴリズムを用いて実装することが望ましい。

発展課題

以下で設定される複数のシーンでエージェントを動作させること。得点に応じて成績を評価する。レポートに書かれた実装のアイデアや記述の詳細度、具体性、発展性についても評価の対象に加える。エージェントはシーンごとに違う実装のものでよいが、同一であれば評価対象に加える。

(1) `ch.idsia.scenarios.MainTask4_2.java`

(2) `ch.idsia.scenarios.MainTask4_3.java`