HAL Command Line Interface Options

RYAN SCHENCK

https://github.com/rschenck/CommandLineHAL

HAL Primary Workflow

Code Development

↓

Execution

↓

Analysis

HAL Workflow

Code Development

- IntelliJ Development IDE
- Base model
- Parameters necessary
- What outputs are needed for analysis?
- Requires prior knowledge of execution requirements

HAL Workflow

Code Development → Execution

- IntelliJ Development IDE
- Base model
- Parameters necessary
- What outputs are needed for analysis?
- Requires prior knowledge of execution requirements

- Executed:
  - Locally?
  - Externally?
  - Locally + Externally?
- Computational limitations?
- JVM arguments necessary?

HAL Workflow

Code Development → Execution → Analysis

- IntelliJ Development IDE
- Base model
- Parameters necessary
- What outputs are needed for analysis?
- Requires prior knowledge of execution requirements

- Executed:
  - Locally?
  - Externally?
  - Locally + Externally?
- Computational limitations?
- JVM arguments necessary?

- Integration through third party packages (like what Maxi showed)
- Post-hoc analysis using Python, R, excel, Matlab
- I reiterate *a priori* knowledge of outputs

HAL Workflow

Code Development → Execution → Analysis

- IntelliJ Development IDE
- Base model
- Parameters necessary
- What outputs are needed for analysis?
- Requires prior knowledge of execution requirements

- Executed:
  - Locally?
  - Externally?
  - Locally + Externally?
- Computational limitations?
- JVM arguments necessary?

- Integration through third party packages (like what Maxi showed)
- Post-hoc analysis using Python, R, excel, Matlab
- I reiterate *a priori* knowledge of outputs

HAL Workflow

Code Development → Execution → Analysis

- IntelliJ Development IDE
- Base model
- Parameters necessary
- What outputs are needed for analysis?
- Requires prior knowledge of execution requirements

- Executed:
  - Locally?
  - Externally?
  - Locally + Externally?
- Computational limitations?
- JVM arguments necessary?

- Integration through third party packages (like what Maxi showed)
- Post-hoc analysis using Python, R, excel, Matlab
- I reiterate *a priori* knowledge of outputs

Execution

- Options for execution:
  - Hard coding parameters for execution
  - Manual command line parsing
  - Command line interfaces (dynamic command line parsing)

Hard coding parameters for execution

- Parameters are housed within the code itself and are not changed.
- The most simple way of performing model execution
- This is the example “BirthDeath.java” on the HAL GitHub.
- It is inflexible as is

```
public class BirthDeath extends AgentGrid2D<Cell> {
    int BLACK=RGB(0,0,0);
    double DEATH_PROB=0.01;
    double BIRTH_PROB=0.2;
    Rand rn=new Rand();
    int[]mooreHood=MooreHood(false);
    int color;
    public BirthDeath(int x, int y,int color) {
        super(x, y, Cell.class);
        this.color=color;
    }
    public void Setup(double rad){
        int[]coords= CircleHood(true,rad);
        int nCoords= MapHood(coords,xDim/2,yDim/2);
        for (int i = 0; i < nCoords ; i++) {
            NewAgentSQ(coords[i]).color=color;
        }
    }
    public void Step() {
        for (Cell c : this) {
            c.Step();
        }
        CleanAgents();
        ShuffleAgents(rn);
    }
    public void Draw(GridWindow vis){
        for (int i = 0; i < vis.length; i++) {
            Cell c = GetAgent(i);
            vis.SetPix(i, c == null ? BLACK : c.color);
        }
    }


    public static void main(String[] args) {
        BirthDeath t=new BirthDeath(100,100, Util.RED);
        GridWindow win=new GridWindow(100,100,10);
        t.Setup(10);
        for (int i = 0; i < 100000; i++) {
            win.TickPause(10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

Hard coding pros and cons

Pros

- Quick
- Easy
- Least amount of code

Cons

- Can’t execute with parameter sets
- Not good for external execution
- Not robust to testing

Manual command line parsing

- Method commonly used by Maxi and Jill
- Provides a way to sweep through parameters
- Gets around the problem of hard coding
- Compiles all your parameters into a single location or possibly class
- Example to the right is Maxi's code. You see a list of all of the parameters with "Helper variables."
  - This is great as it keeps the parameters up front for those wanting to reuse

```
/* ================================================================
 * --- Parameters ----
 * ================================================================ */
int xDim = 100; // x-Dimensions of the grid
int yDim = 100; // y-Dimensions of the grid
int[] initPopSizeArr = {0,0}; // Initial population sizes
double divisionRate_S = .027;
double divisionRate_R = divisionRate_S;
double deathRate_S = 0;
double deathRate_R = 0;
double movementRate = 0.;
double drugKillProportion = 0.75; // Corresponds to 1.5 from ODE
String initialSeedingType = "random"; // How cells are initially distributed. Options: "random", "separation1d", "separation2d", "circle"
int initialSeedingDistance = 10; // Distance which cells are seed apart if seeded using "separation_x", or radius of circle if seeded as "circle"
Boolean compareToMTD = true; // Compare AT to MTD. If false simulate only the single treatment given by atThreshold.
double atThreshold = 0.5; // Relative size reduction when treatment is withdrawn under AT
Boolean simulateSpecificSchedule = false; // Whether to simulate a specific pre-defined Tx schedule
double[][] treatmentScheduleList = null; // Treatment schedule to simulate
Boolean predictIntermittentTherapyOutcome = false; // Whether to predict the outcome according to the Bruchovsky et al (2006) intermittent schedule
double initialPSA = 1; // Baseline PSA value. Used for determining normal and elevated ranges during intermittent schedule
int weeksOnTreatment = 0; // Weeks on treatment prior to start of intermittent therapy prediction
int[] previousFourMeasurements = new int[] {0,0,0,0}; // If patient had been receiving therapy, these are their last four measurements
Boolean simulateCT = false; // Whether to simulate IMT or CT
double proProlifFac = 0; // Factor by which growth rate is increased during off-treatment phases
double proProlifBalance = 1; // Relative effect of pro proliferation treatment on resistant cells
double proTurnoverFac = 0; // Relative increase in death rate in 'off-treatment' phase due to low-dose chemo
double proTurnoverBalance = 1; // Relative effect of pro turnover treatment on resistant cells
double[][] proTurnoverSchedule=null; // Schedule of pro-turnover drug. Used to obtain controls for pro-turnover experiments.
double dt = 1.;
double tEnd = 10000; // End time
double[] paramArr;
int nReplicates = 1000;
String outDir = "./tmp/";

// Helper variables
OnLatticeCA myModel = new OnLatticeCA();
String outFName;
String txName;
double initialSizeProp=-1;
int initialSize=-1;
double rFrac=-1;
double cost=-1;
Boolean profilingMode=true;
Boolean terminateAtProgression = true;
int seed=-1;
String imageOutDir="";
int imageFreq=10;
```

Assigning variables from command line

- You have your jarfile. You've created it using the IntelliJ method outlined in the HAL manual. How do you read in variables
- Two methods!
  - Method 1 always set every variable (Jill)
  - Method 2 create a manual command line parser (Maxi)

Method 1 and 2: Step 1

```
public class BirthDeath extends AgentGrid2D<Cell> {
    int BLACK=RGB(0,0,0);
    double DEATH_PROB=0.01;
    double BIRTH_PROB=0.2;
    Rand rn=new Rand();
    int[]mooreHood=MooreHood(false);
    int color;
    public BirthDeath(int x, int y,int color) {
        super(x, y, Cell.class);
        this.color=color;
    }
    public void Setup(double rad){
        int[]coords= CircleHood(true,rad);
        int nCoords= MapHood(coords,xDim/2,yDim/2);
        for (int i = 0; i < nCoords ; i++) {
            NewAgentSQ(coords[i]).color=color;
        }
    }
    public void Step() {
        for (Cell c : this) {
            c.Step();
        }
        CleanAgents();
        ShuffleAgents(rn);
    }
    public void Draw(GridWindow vis){
        for (int i = 0; i < vis.length; i++) {
            Cell c = GetAgent(i);
            vis.SetPix(i, c == null ? BLACK : c.color);
        }
    }


    public static void main(String[] args) {
        BirthDeath t=new BirthDeath(100,100, Util.RED);
        GridWindow win=new GridWindow(100,100,10);
        t.Setup(10);
        for (int i = 0; i < 100000; i++) {
            win.TickPause(10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

- Get parameters into single class and put all together
- Replace all the parameters with the parameter variable
- Note: This is the same as the hard coded, but now in a centralized location

```
class PARAMS {
    public static int DIM=100;
    public static int RUNTIME=100000;
    public static double DEATH_PROB=0.01;
    public static double BIRTH_PROB=0.2;
}

public class Example {

    public static void main(String[] args) {

        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

PARAMS. VARIABLE

Method 1: Step 2

```
class PARAMS {
    public static int DIM=100;
    public static int RUNTIME=100000;
    public static double DEATH_PROB=0.01;
    public static double BIRTH_PROB=0.2;
}

public class Example {

    public static void main(String[] args) {

        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

PARAMS.VARIABLE

- Parse and assign parameters in order
- If no parameters are assigned, defaults are used.

```
class PARAMS {
    public static int DIM=100;
    public static int RUNTIME=100000;
    public static double DEATH_PROB=0.01;
    public static double BIRTH_PROB=0.2;
}

public class Example {

    public static void main(String[] args) {

        if(args.length > 0){//if arguments, get argments
            PARAMS.DIM = Integer.parseInt(args[0]);
            PARAMS.RUNTIME = Integer.parseInt(args[1]);
            PARAMS.DEATH_PROB = Double.parseDouble(args[2]);
            PARAMS.DEATH_PROB = Double.parseDouble(args[3]);
        } else{//no arguments
            System.out.println("No arguments");
        }

        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

Method 2: Step 2

```
class PARAMS {
    public static int DIM=100;
    public static int RUNTIME=100000;
    public static double DEATH_PROB=0.01;
    public static double BIRTH_PROB=0.2;
}

public class Example {

    public static void main(String[] args) {

        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

PARAMS.VARIABLE

- Parse and assign parameters based on “tag”
- Can specify zero or 1+ arguments and it will handle itself

```
public class Example {

    public static void main(String[] args) {

        for (int i = 0; i < args.length; i+=2) {
            if (args[i].equalsIgnoreCase( anotherString: "-dim")) {
                PARAMS.DIM = Integer.parseInt(args[i + 1]);
            }
            if (args[i].equalsIgnoreCase( anotherString: "-t")) {
                PARAMS.RUNTIME = Integer.parseInt(args[i + 1]);
            }
            if (args[i].equalsIgnoreCase( anotherString: "-death_prob")) {
                PARAMS.DEATH_PROB = Double.parseDouble(args[i + 1]);
            }
            if (args[i].equalsIgnoreCase( anotherString: "-birth_prob")) {
                PARAMS.DEATH_PROB = Double.parseDouble(args[i + 1]);
            }
        }

        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

Manual command line parsing

Pros

- Easy
- Specify every variable reduces chance of error (method 1)
- Easiest way to be dynamic (method 2)

Cons

- Verbose
- No help or version numbers.
- Not easily shared.

Command line interface

- Primarily for deployable execution
- Utilizes picocli (https://picocli.info)
- Small and powerful library to do what method 2 does in the previous example + extras
- Get a jar file or use maven to link picocli to your project using ‘project structure and ‘library’

picocli

Command line interface

- What drew me to using Picocli?
  - I’m used to using software in this way on the command line from bioinformatics tools (+ others)
- All of the features from the previous methods, but a lot of extras and reduced verbosity.
- Provides you with versioning and a help menu for collaborations and/or yourself
- Shown is an example of Gattaca’s example model that has three underlying models each with a set of available to specify parameters
  - You get this help menu from running:
    - $ java -jar Gattaca.jar -- help

```
Usage:

Gattaca

Gattaca Models [-hV] [-b=<birth_rate>] [-d=<death_rate>] [-i=<dimension>]
               [-k=<popSize>] [-m=<theModel>] [-o=<outdir>] [-t=<sim_time>]
               [-u=<mutrate>]

Description:

Gattaca Example Models

Options:
  -b, --birth_rate=<birth_rate>
                            Birth Rate.
                              Default: 1.0
  -d, --death_rate=<death_rate>
                            Death Rate.
                              Default: 0.1
  -h, --help                Show this help message and exit.
  -i, --dimension=<dimension>
                            Dimension of the simulation. 0d, 2d, 3d. Defualt 0d
                              Default: 2d
  -k, --carrying_cap=<popSize>
                            Carrying capacity, k. Defines the grid size as well.
                              Default: 15625
  -m, --model=<theModel>    Either Tissue for fully seeded or Cancer for
                              starting from a single cell.
                              Default: Cancer
  -o, --output_dir=<outdir> Output directory name. Does not have to exist, it
                              will be created.
                              Default:
                              /Users/4466451/Documents/Projects/Gattaca/GattacaE
                              xample/0d/
  -t, --time=<sim_time>     Time for simulation.
                              Default: 5000
  -u, --mutrate=<mutrate>   Mutation rate as supplied from Gattaca.
                              Default: 10-9
  -V, --version             Print version information and exit.
```

Step 1: Setup variables

- Parameters start out in their own class in the similar way that it's been done in the other examples.
- Parameters are specified in the appropriate locations while referencing this parameter class
- Add a picocli command line argument
  - There's also arrays, booleans, anything else you would want here.

```
@CommandLine.Option(names={"-s","--size"}, description="Size to use for x and y dimensions.") int dim=PARAMS.DIM;
```

It's an argument

Flags (either full name or shorthand)

What houses the variable

Step 1b: Optional interface settings

- We have the power, make it look good and customize your information shown to you/users

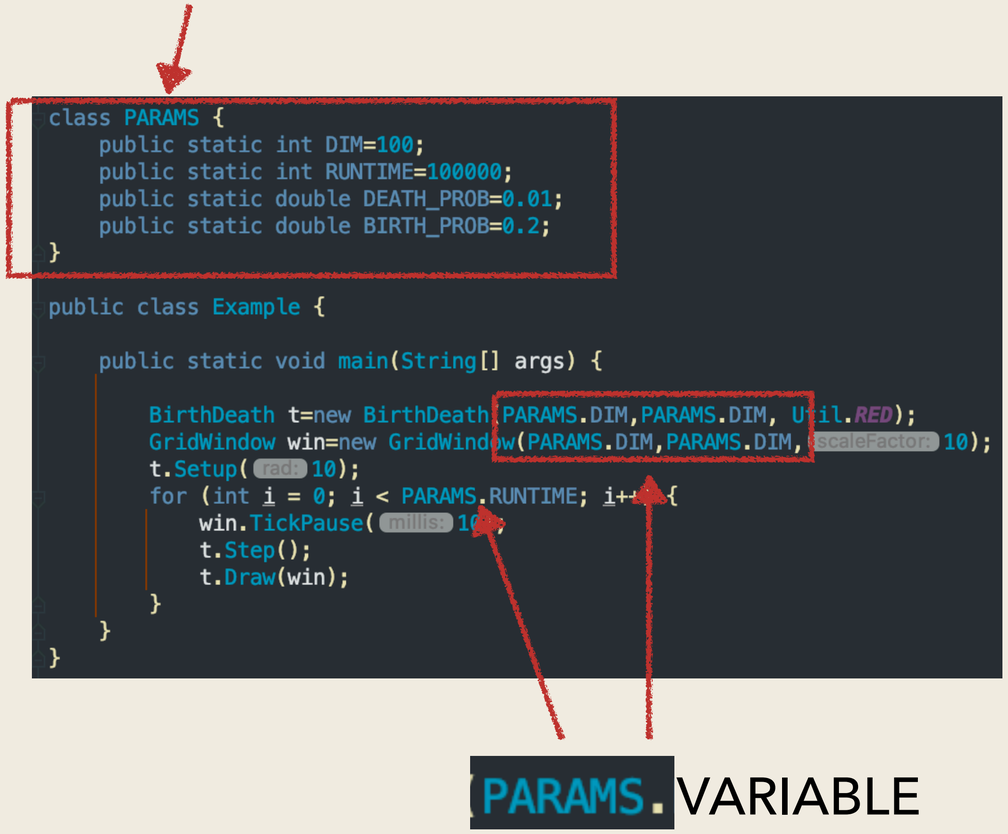
Command line interface options,
colors, display, text, version, verbosity,
etc.

```
@CommandLine.Command(name = "Example model",
        header="Example",
        mixinStandardHelpOptions = true,
        showDefaultValues = true,
        sortOptions = true,
        headerHeading = "@|bold,underline,green Usage|@:%n%n",
        synopsisHeading = "%n@|red Put as many freaking|@:%n %n@|underline,bold,green words as you want here to give in detail information!|@:%n%n",
        descriptionHeading = "%n@|bold,underline,red Description|@:%n%n",
        parameterListHeading = "%n@|bold,underline,blue Parameters|@:%n",
        optionListHeading = "%n@|bold,underline,red Options|@:%n",
        version = "v3.14",
        description = "Example Command Line Interface Model")
public class Example implements Runnable {
    @CommandLine.Option(names={"-s","--size"}, description="Size to use for x and y dimensions") int dim=PARAMS.DIM;
    @CommandLine.Option(names={"-t","--time"}, description="Run time. Default") int runtime=PARAMS.RUNTIME ;
    @CommandLine.Option(names={"-d","--death_rate"}, description="Death rate.") double deathProb=PARAMS.DEATH_PROB;
    @CommandLine.Option(names={"-b","--birth_rate"}, description="Birth rate.") double birthProb=PARAMS.BIRTH_PROB;
```

Arguments

```
Usage:

Example

Put as many freaking:

words as you want here to give in detail information!:

Example model [-hV] [-b=<birthProb>] [-d=<deathProb>] [-s=<dim>] [-t=<runtime>]

Description:

Example Command Line Interface Model

Options:
  -b, --birth_rate=<birthProb>
                            Birth rate.
                              Default: 0.2
  -d, --death_rate=<deathProb>
                            Death rate.
                              Default: 0.01
  -h, --help                Show this help message and exit.
  -s, --size=<dim>          Size to use for x and y dimensions
                              Default: 100
  -t, --time=<runtime>      Run time. Default
                              Default: 100000
  -V, --version             Print version information and exit.
```

Step 2: Setup the code



PARAMS.VARIABLE

- Parameters start out in their own class in the similar way that it's been done in the other examples.
- Parameters are specified in the appropriate locations while referencing this parameter class
- Implement a "run" class and make class Example "runnable"

```
public class Example {
```

```
public class Example implements Runnable {
```

+

```
public void run() {

}
```

Step 3: Add options and reorganize

```java
public class Example implements Runnable {
    @CommandLine.Option(names={"-b","--birth_rate"}, description="Birth Rate.") int dim=PARAMS.DIM;
    @CommandLine.Option(names={"-d","--death_rate"}, description="Death Rate.") int runtime=PARAMS.RUNTIME ;
    @CommandLine.Option(names={"-t","--time"}, description="Time for simulation.") double deathProb=PARAMS.DEATH_PROB;
    @CommandLine.Option(names={"-k","--carrying_cap"}, description="Carrying capacity, k. Defines the grid size as well.") double birthProb=PARAMS.BIRTH_PROB;

    public void run(){
        PARAMS.DIM = dim;
        PARAMS.RUNTIME = runtime;
        PARAMS.DEATH_PROB = deathProb;
        PARAMS.BIRTH_PROB = birthProb;

        try {
            RunModel();
        } catch (Exception e){
            System.out.println("IOException for RunModel()");
        }
    }

    public static void main(String[] args) {
        CommandLine.run(new Example(), args);
    }

    public static void RunModel(){
        BirthDeath t=new BirthDeath(PARAMS.DIM,PARAMS.DIM, Util.RED);
        GridWindow win=new GridWindow(PARAMS.DIM,PARAMS.DIM, scaleFactor: 10);
        t.Setup( rad: 10);
        for (int i = 0; i < PARAMS.RUNTIME; i++) {
            win.TickPause( millis: 10);
            t.Step();
            t.Draw(win);
        }
    }
}
```

Setup your environment and Run the model

The magic in PSVM

RunModel is now the model execution function

- You now have a command line interface setup to parse any and all commands with exceptions should any arguments be incorrect

Trick: This won't run in the IDE anymore unless...

```
Unmatched arguments from index 0: '100', '10000', '0.01', '0.2'
Usage:

Example

Put as many freaking:

words as you want here to give in detail information!:

Example model [-hV] [-b=<birthProb>] [-d=<deathProb>] [-s=<dim>] [-t=<runtime>]

Description:

Example Command Line Interface Model

Options:
  -b, --birth_rate=<birthProb>
                          Birth rate.
                            Default: 0.2
  -d, --death_rate=<deathProb>
                          Death rate.
                            Default: 0.01
  -h, --help              Show this help message and exit.
  -s, --size=<dim>        Size to use for x and y dimensions
                            Default: 100
  -t, --time=<runtime>    Run time. Default
                            Default: 100000
  -V, --version           Print version information and exit.
```

- If you try to execute it, it will just show you the help menu and exit
- Add a boolean for jar file and only package with that set to true
  - For myself, I call this 'development mode'

```
public static void main(String[] args) {
    if(PARAMS.JAR){
        CommandLine.run(new Example(), args);
    } else {
        RunModel();
    }
}
```

Dynamic command line parsing - Picocli

Pros

- Makes your code very easy for others to use
- A lot of flexibility
- Powerful
- Least verbose for projects with many variables/options

Cons

- Minor learning curve
- Not suited for small models
- Likewise if only executing your code yourself and locally it likely isn't needed.

Run any of these approaches after creating a jar file any way you currently do on the cluster or locally

https://github.com/rschenck/CommandLineHAL