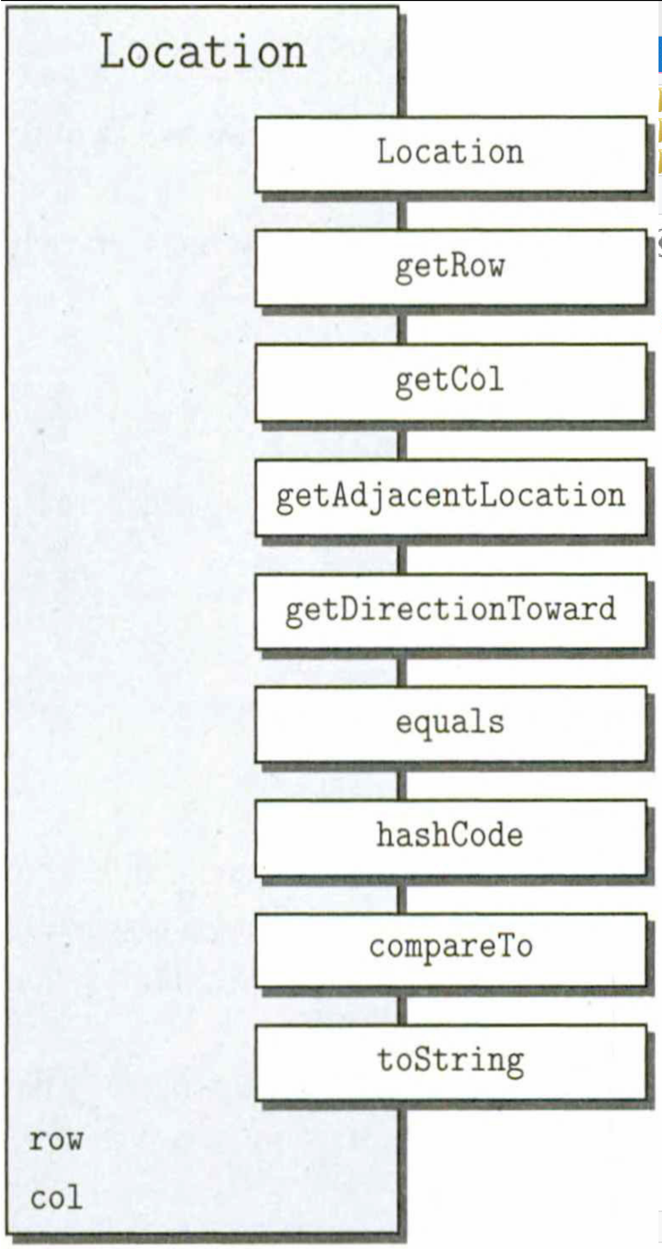
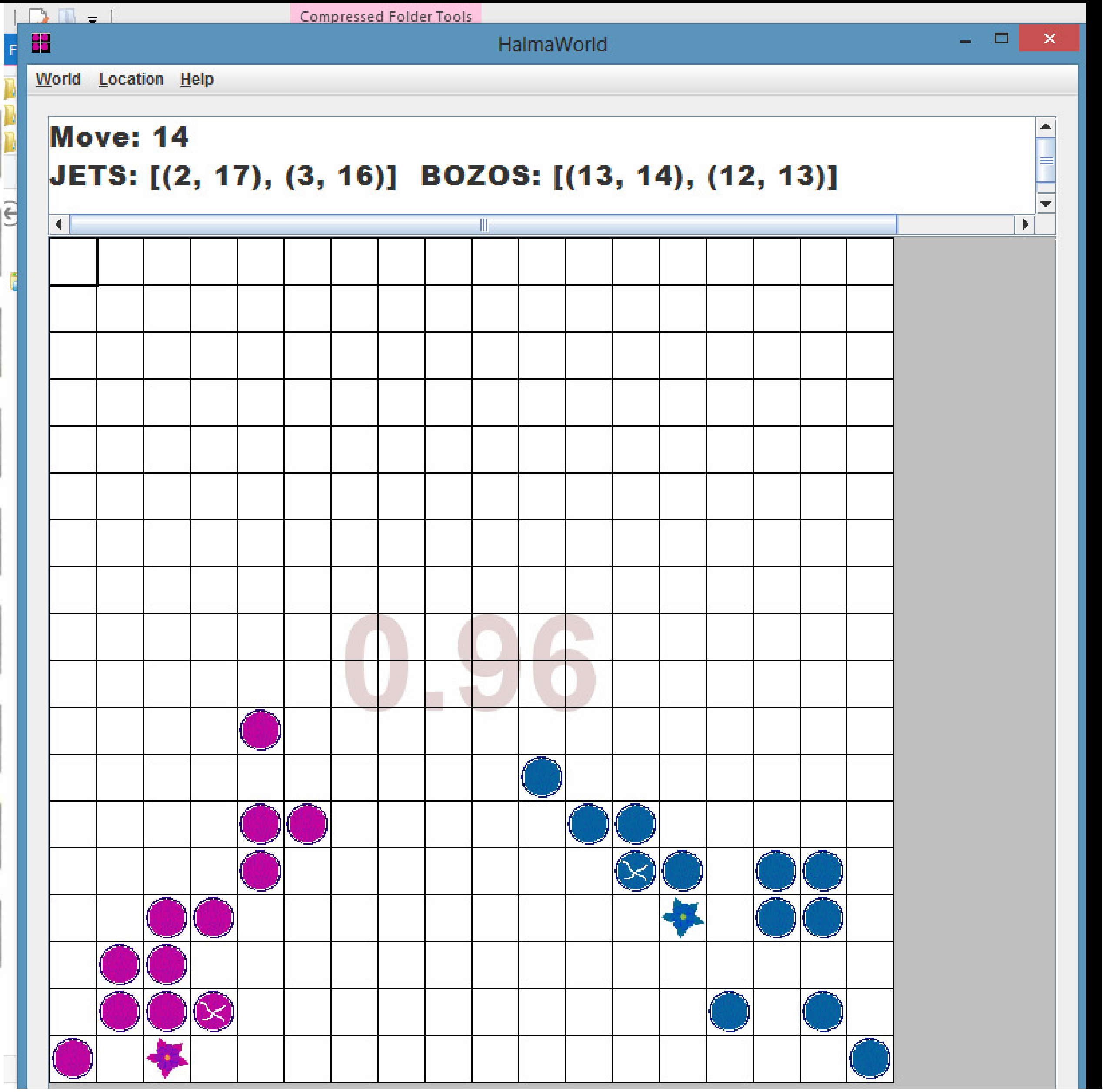
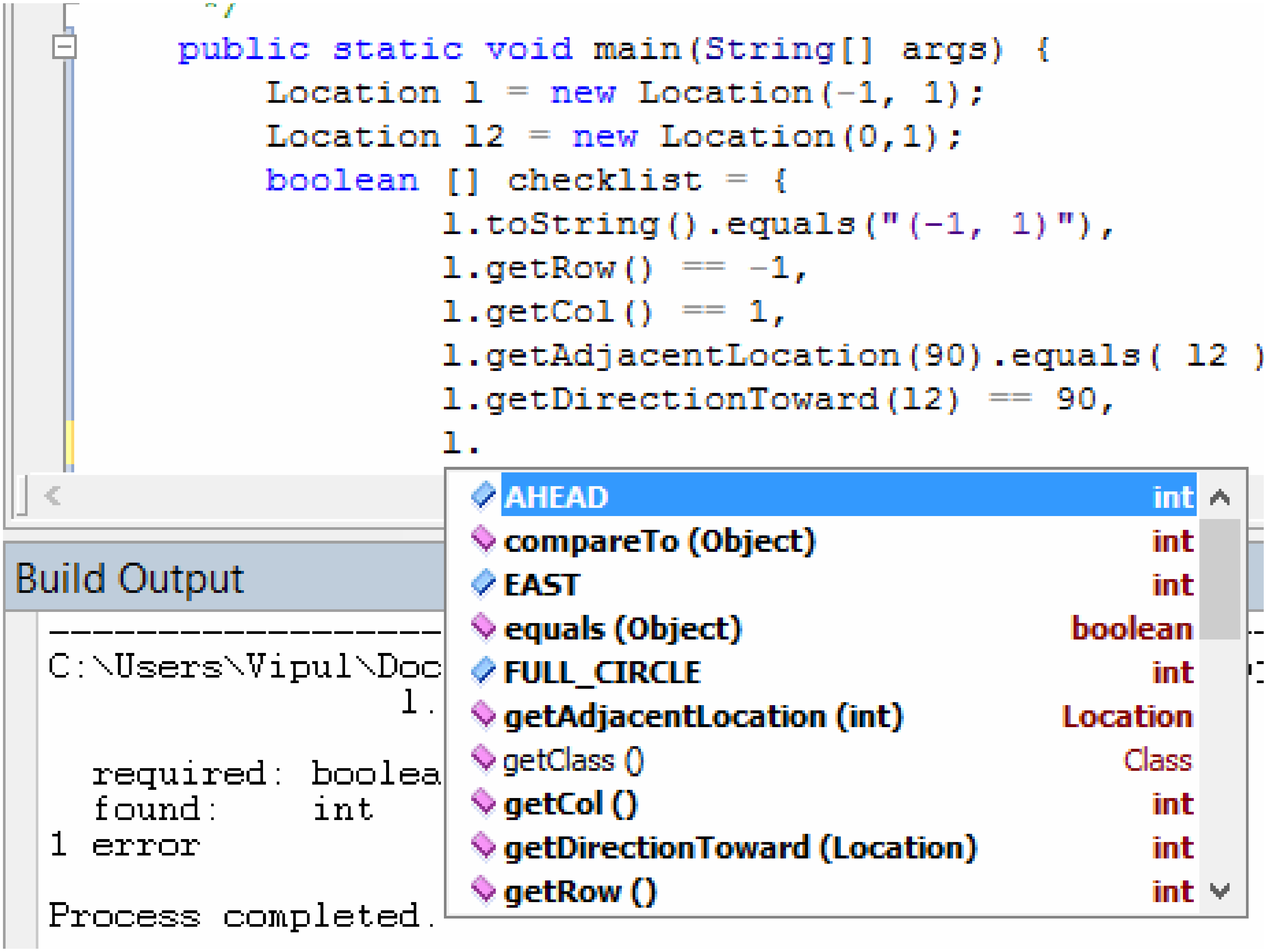
HalmaWorld & Location





Move: 14
JETS: [(2, 17), (3, 16)] BOZOS: [(13, 14), (12, 13)]



```
    public static void main(String[] args) {
        Location l = new Location(-1, 1);
        Location l2 = new Location(0,1);
        boolean [] checklist = {
                l.toString().equals("(-1, 1)"),
                l.getRow() == -1,
                l.getCol() == 1,
                l.getAdjacentLocation(90).equals( l2 )
                l.getDirectionToward(l2) == 90,
                l.
```

```
AHEAD                                  int
compareTo (Object)                     int
EAST                                   int
equals (Object)                        boolean
FULL_CIRCLE                            int
getAdjacentLocation (int)              Location
getClass ()                            Class
getCol ()                              int
getDirectionToward (Location)          int
getRow ()                              int
```

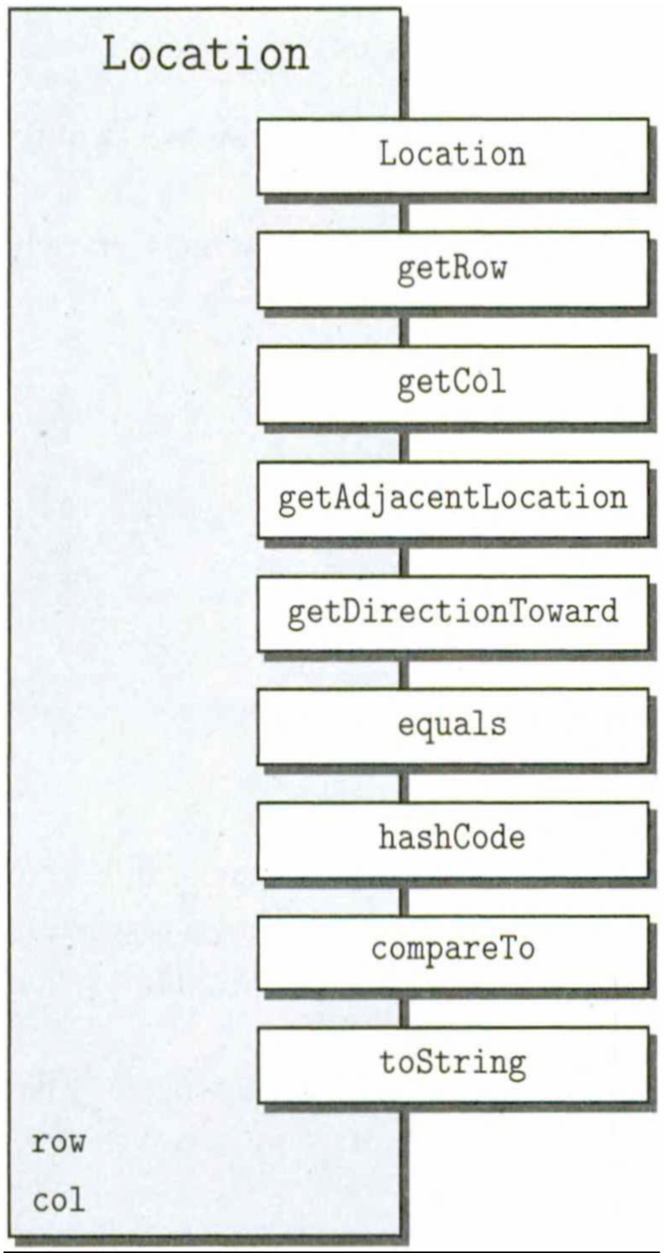
Build Output

```
--------------------
C:\Users\Vipul\Doc
                  l.

  required: boolea
  found:    int
1 error

Process completed.
```

Stepwise Checklist for (3 , 0)

Method	Received	Expected
new Location()	(3,0)	(3,0)
getRow()	3	0
getCol()	0	3
Equals((3,0))	true	true

- Great learning tool
- Important for validation
- Tests against documentation



[redacted]

Methods

```
public Location(int r, int c)
```

Constructs a location with given row and column.

```
public int getRow()
public int getCol()
```

Accessor methods that return the row or column of the `Location`.

```
public Location getAdjacentLocation(int direction)
```

Returns the adjacent location in the compass direction closest to `direction`.

```
public int getDirectionToward(Location target)
```

Returns the direction, rounded to the nearest compass direction, from this location toward a target location.

```
public int hashCode()
```

Generates and returns a hash code for this `Location`.

```
public boolean equals(Object other)
public int compareTo(Object other)
```

These methods are used to compare `Location` objects. Two locations are equal if they have the same row and column values. Ascending order for locations is *row-major*, namely, start at (0, 0) and proceed row by row from left to right. For example, (0, 1) is less than (0, 2) and (1, 8) is less than (3, 0).

```
public String toString()
```

Returns a string representation of this `Location` in the form (row, col).

Automated Checklist for (0 , 0)

```
  public static void main(String[] args) {
      Location loc = new Location(0, 0);
      Location l2 = new Location(0, 1);
//----------------------------------------------------
      boolean [] checklist = {
              loc.toString().equals("(0, 0)"),
              loc.getRow() == 0,
              loc.getCol() == 0,
              loc.getAdjacentLocation(Location.EAST).equals( l2 ),
              loc.getDirectionToward( l2 ) == 90,
              loc.hashCode() != 0,
              loc.compareTo( new Location(0,1) ) == -1,
              loc.compareTo( new Location(100,0) ) == -100,
          };
//-------------------------------------------------------
    for (boolean testResult : checklist)
         System.out.println(testRestult);
```

Automated Checklist for (0 , 0)

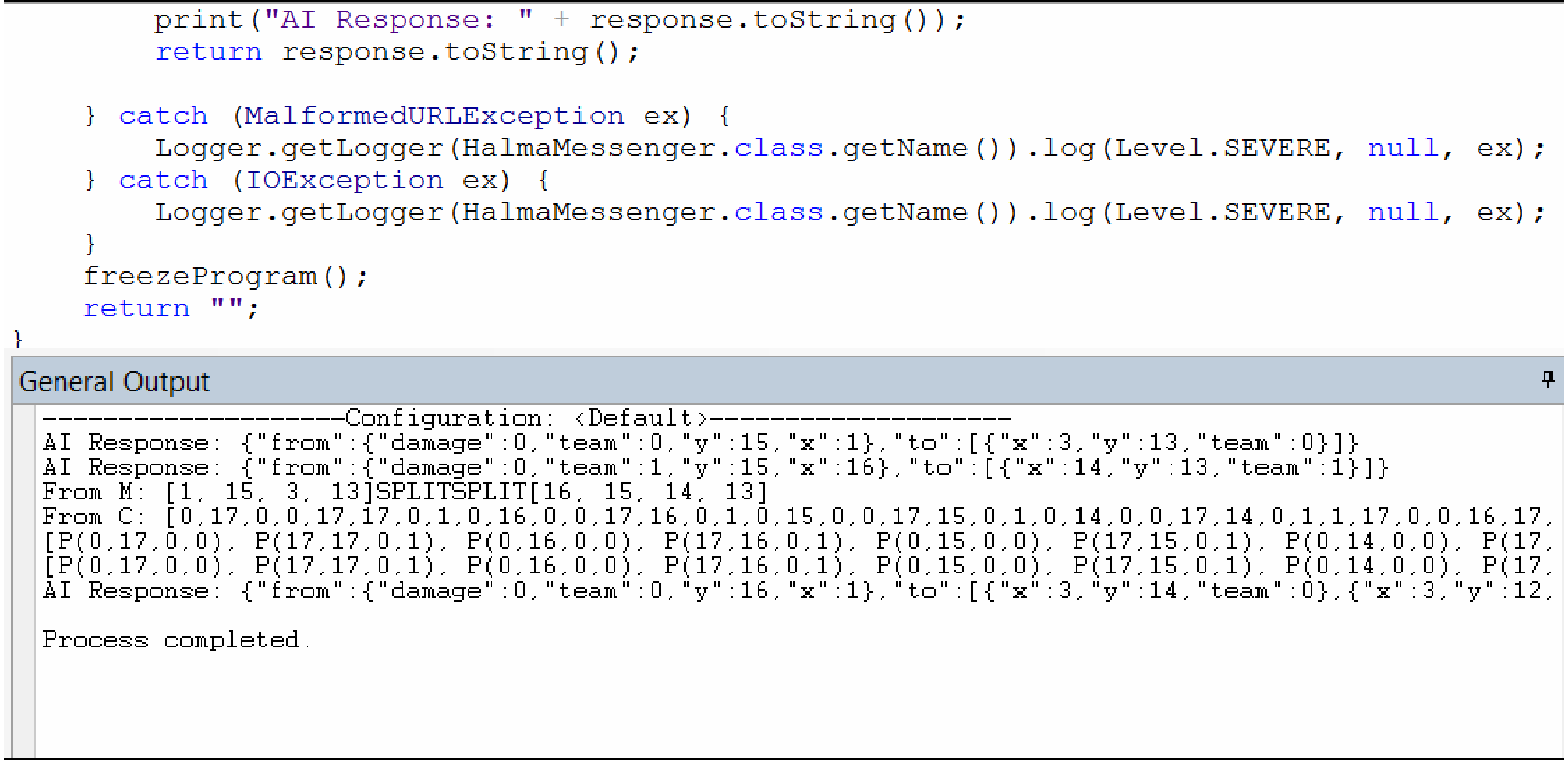
Expected	Received
true	true
true	true
true	true
true	true
true	true
false	false
true	true
→true	→false

this.compareTo(new Location(100,0))

Returns -1

Not -100 as expected

White-Box Checklist Testing



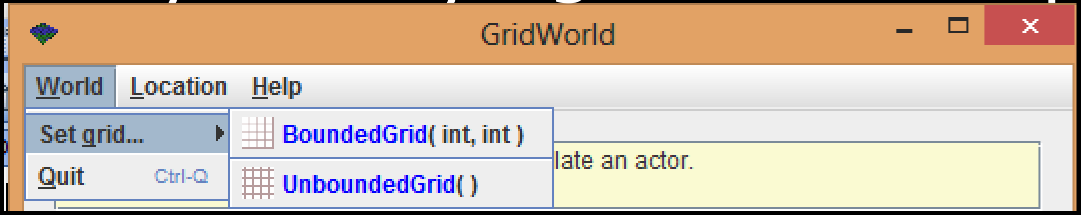
```
        print("AI Response: " + response.toString());
        return response.toString();

    } catch (MalformedURLException ex) {
        Logger.getLogger(HalmaMessenger.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(HalmaMessenger.class.getName()).log(Level.SEVERE, null, ex);
    }
    freezeProgram();
    return "";
}
```

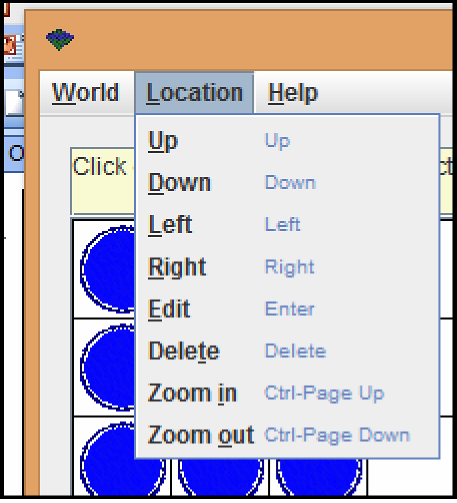
General Output

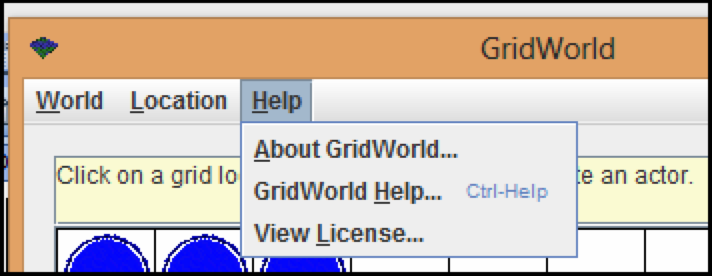
```
--------------------Configuration: <Default>--------------------
AI Response: {"from":{"damage":0,"team":0,"y":15,"x":1},"to":[{"x":3,"y":13,"team":0}]}
AI Response: {"from":{"damage":0,"team":1,"y":15,"x":16},"to":[{"x":14,"y":13,"team":1}]}
From M: [1, 15, 3, 13]SPLITSPLIT[16, 15, 14, 13]
From C: [0,17,0,0,17,17,0,1,0,16,0,0,17,16,0,1,0,15,0,0,17,15,0,1,0,14,0,0,17,14,0,1,1,17,0,0,16,17,
[P(0,17,0,0), P(17,17,0,1), P(0,16,0,0), P(17,16,0,1), P(0,15,0,0), P(17,15,0,1), P(0,14,0,0), P(17,
[P(0,17,0,0), P(17,17,0,1), P(0,16,0,0), P(17,16,0,1), P(0,15,0,0), P(17,15,0,1), P(0,14,0,0), P(17,
AI Response: {"from":{"damage":0,"team":0,"y":16,"x":1},"to":[{"x":3,"y":14,"team":0},{"x":3,"y":12,

Process completed.
```

GridWorld Menu

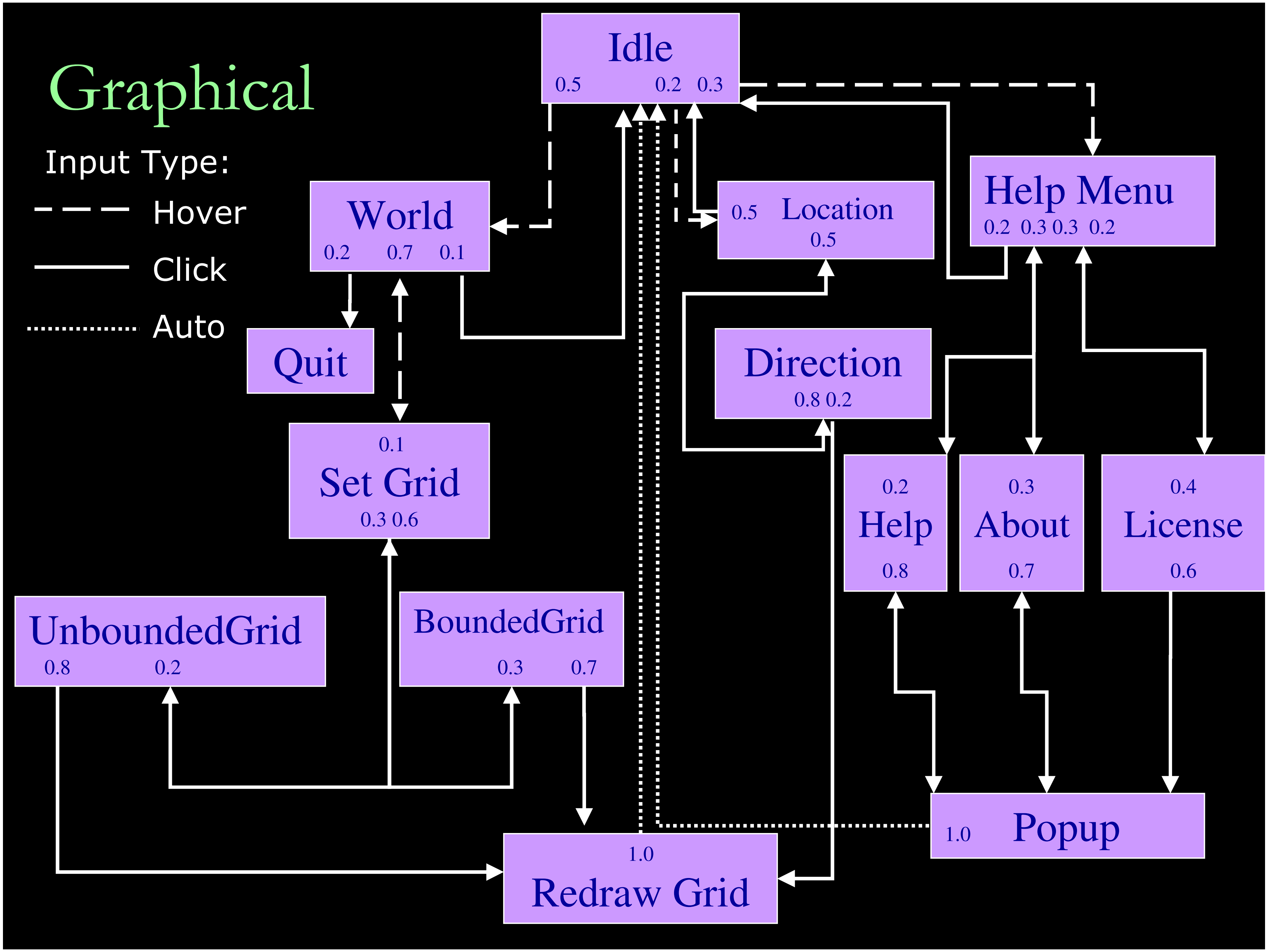
- A UI to develop grid-based games such as chess, checkers, battleship, or pacman.
- Probabilities can either be measured by logging user mouse behavior or estimated by surveying users or expert developers.







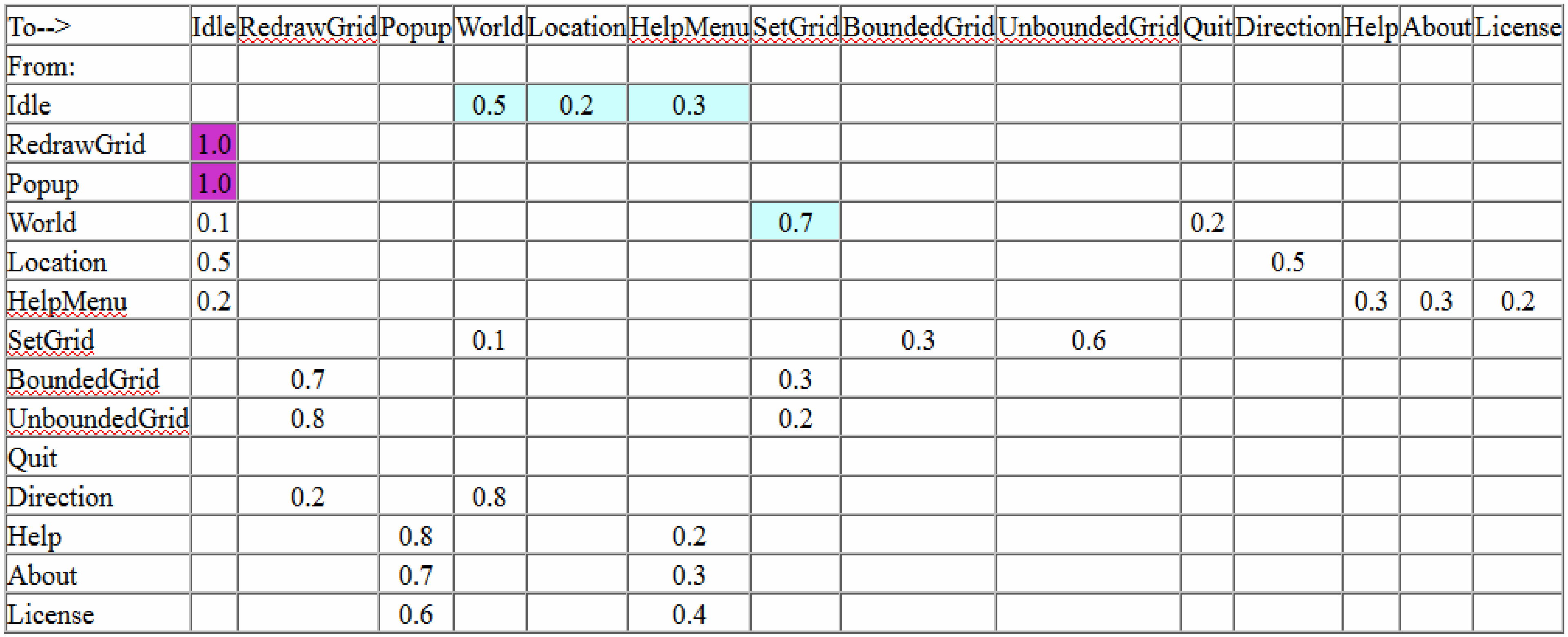
Graphical


Input Type:
Hover
Click
Auto
Idle
0.5 0.2 0.3
World
0.2 0.7 0.1
Quit
Set Grid
0.1
0.3 0.6
Location
0.5
0.5
Direction
0.8 0.2
Help Menu
0.2 0.3 0.3 0.2
Help
0.2
0.8
About
0.3
0.7
License
0.4
0.6
UnboundedGrid
0.8 0.2
BoundedGrid
0.3 0.7
Redraw Grid
1.0
Popup
1.0

Tabular (Matrix)

Input type:

White = Click   Sky = Hover   Purple = Auto

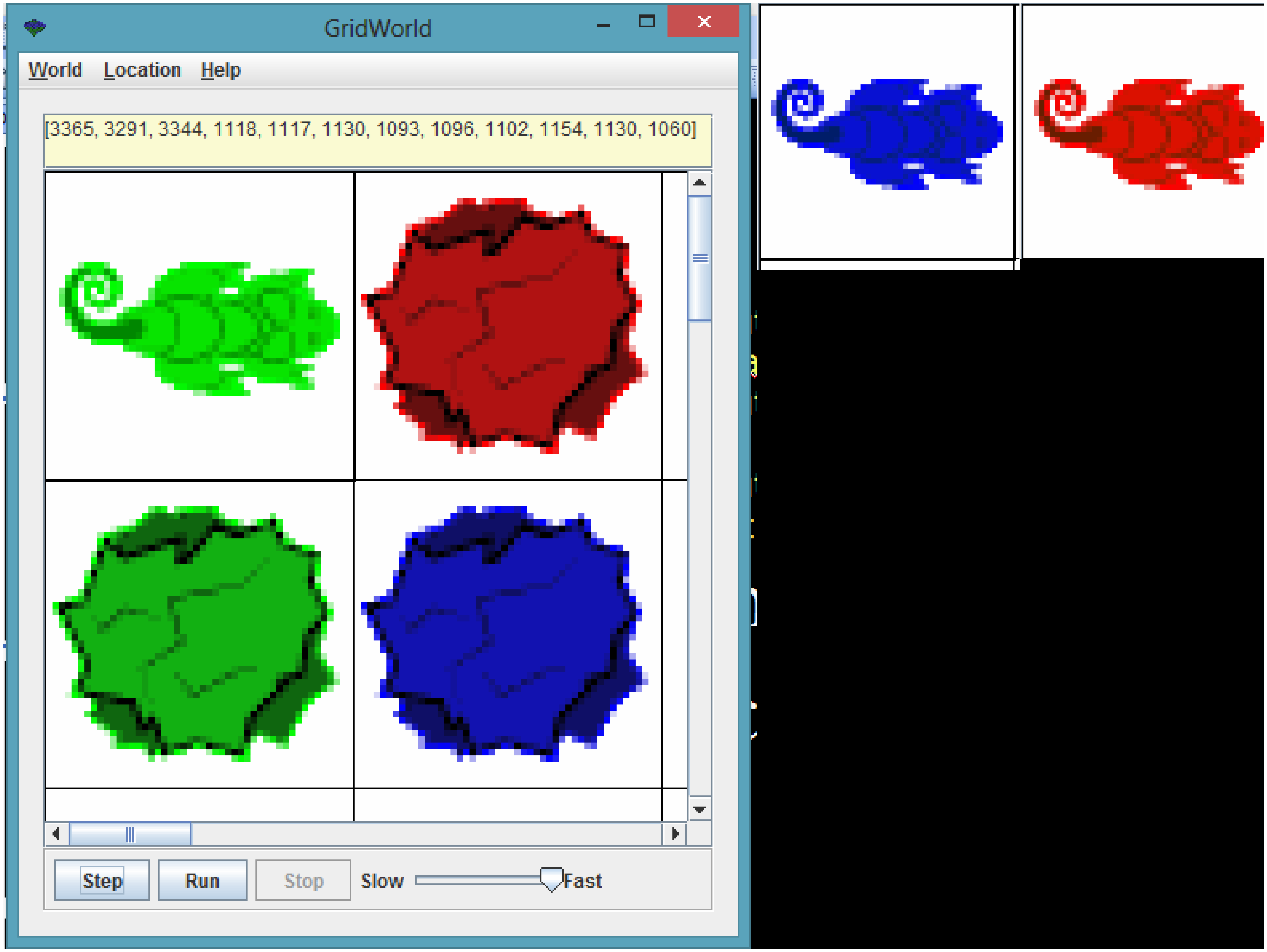


To-->	Idle	RedrawGrid	Popup	World	Location	HelpMenu	SetGrid	BoundedGrid	UnboundedGrid	Quit	Direction	Help	About	License
From:														
Idle				0.5	0.2	0.3								
RedrawGrid	1.0													
Popup	1.0													
World	0.1						0.7			0.2				
Location	0.5										0.5			
HelpMenu	0.2											0.3	0.3	0.2
SetGrid				0.1				0.3	0.6					
BoundedGrid		0.7					0.3							
UnboundedGrid		0.8					0.2							
Quit														
Direction		0.2		0.8										
Help			0.8			0.2								
About			0.7			0.3								
License			0.6			0.4								

Flat Musa OP (Threshold P = 0.050)

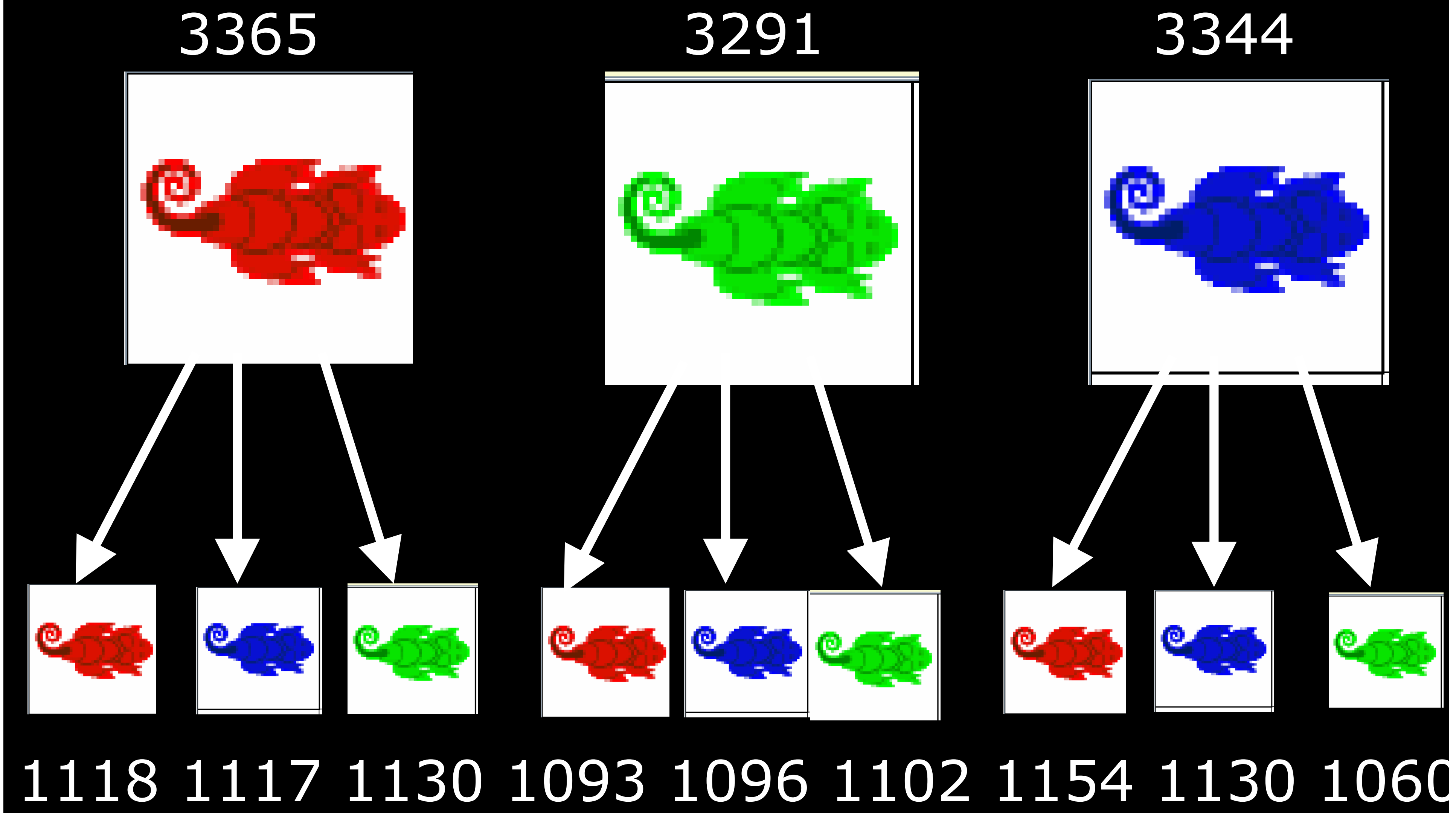
Redraw/Popup Path	Probability
Idle/World/SetGrid/BoundedGrid/RedrawGrid	0.5 * 0.7 * 0.6 * 0.7 = 0.147
Idle/World/SetGrid/UnboundedGrid/RedrawGrid	0.5 * 0.7 * 0.3 * 0.8 = 0.084
Idle/HelpMenu/Help/Help/Popup	0.3 * 0.3 * 0.8 = 0.072
Idle/HelpMenu/Help/About/Popup	0.3 * 0.3 * 0.8 = 0.063

- The Redraw/Popup paths are most meaningful practically.
- The path determines the type of grid drawn or popup displayed.

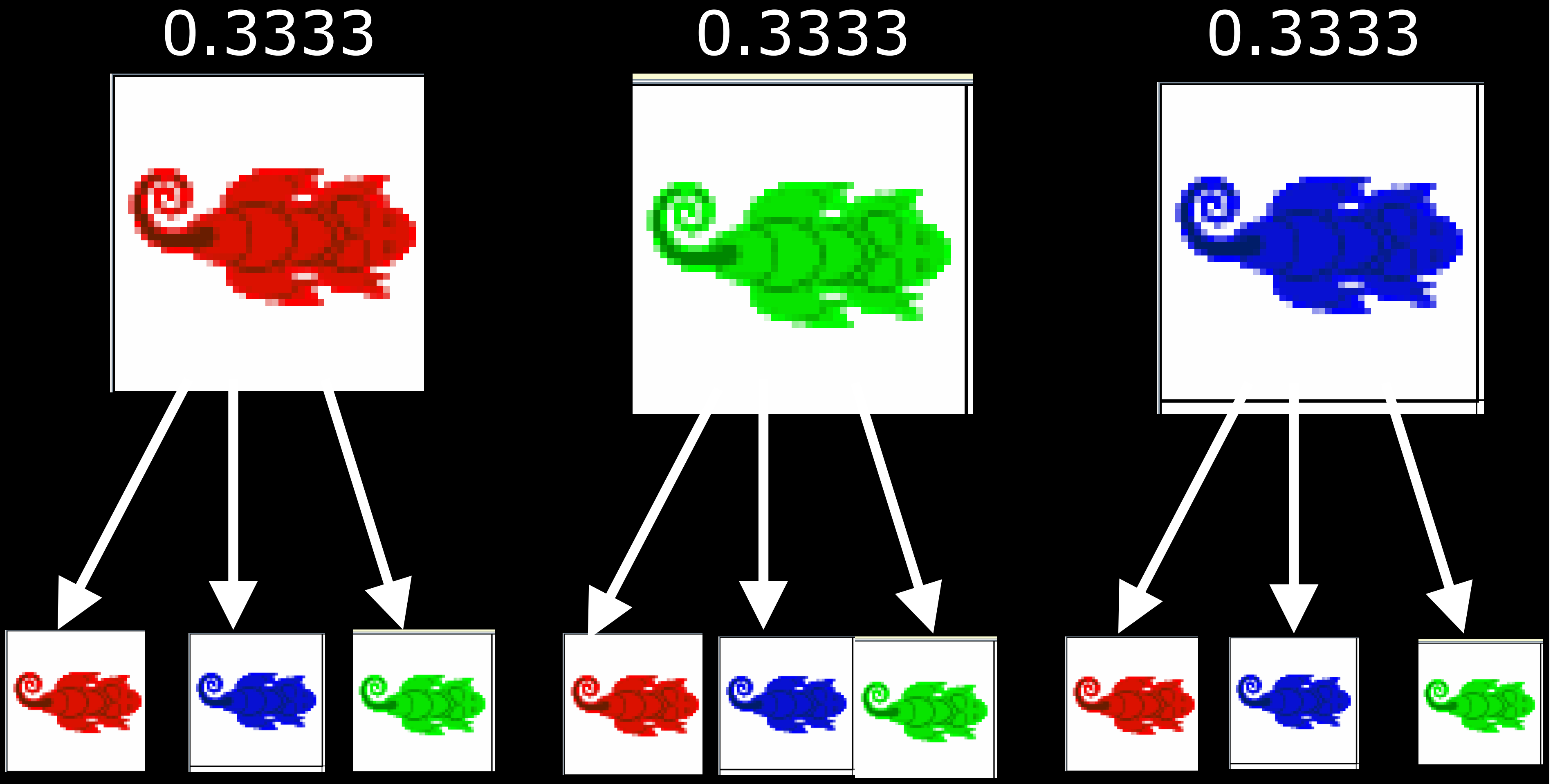


[3365, 3291, 3344, 1118, 1117, 1130, 1093, 1096, 1102, 1154, 1130, 1060]

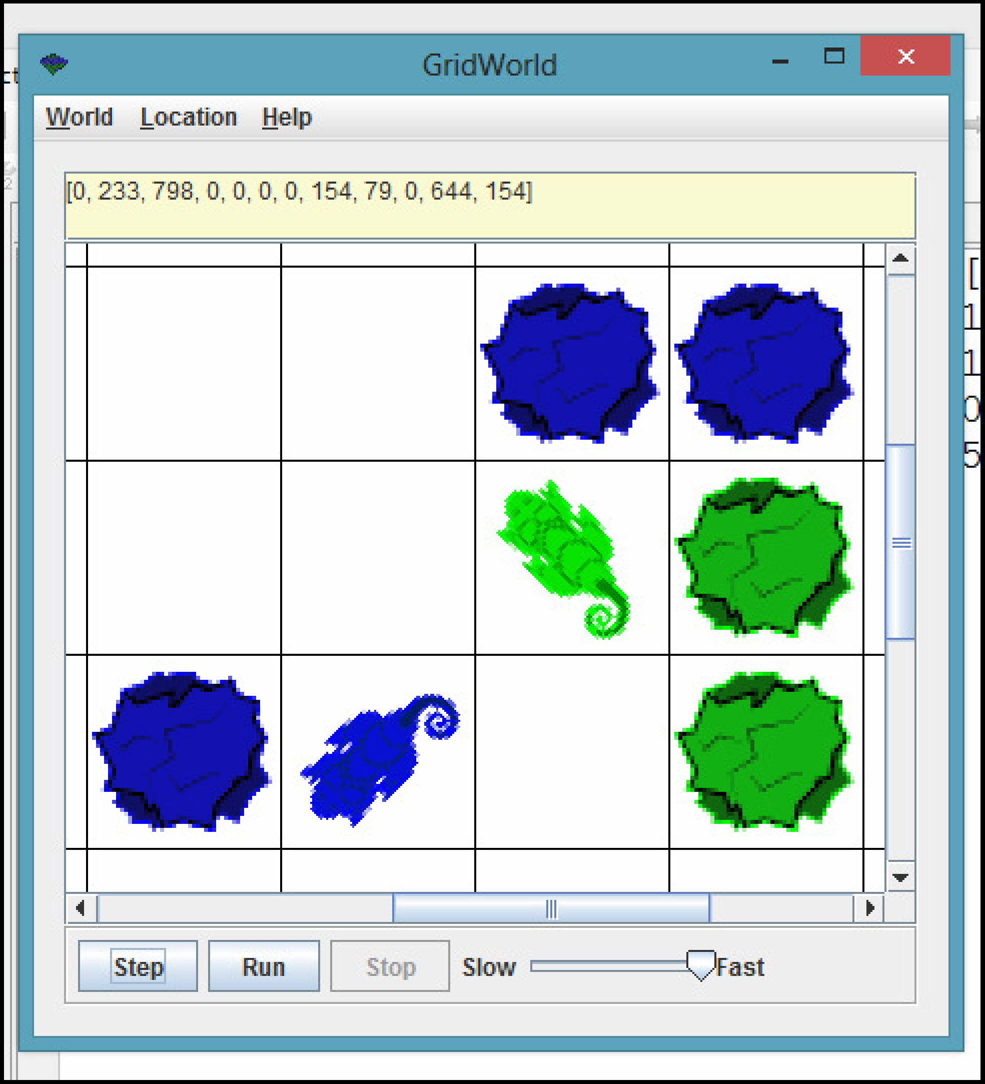
10K Experimental Musa-2 OP



Theoretical Musa-2 OP

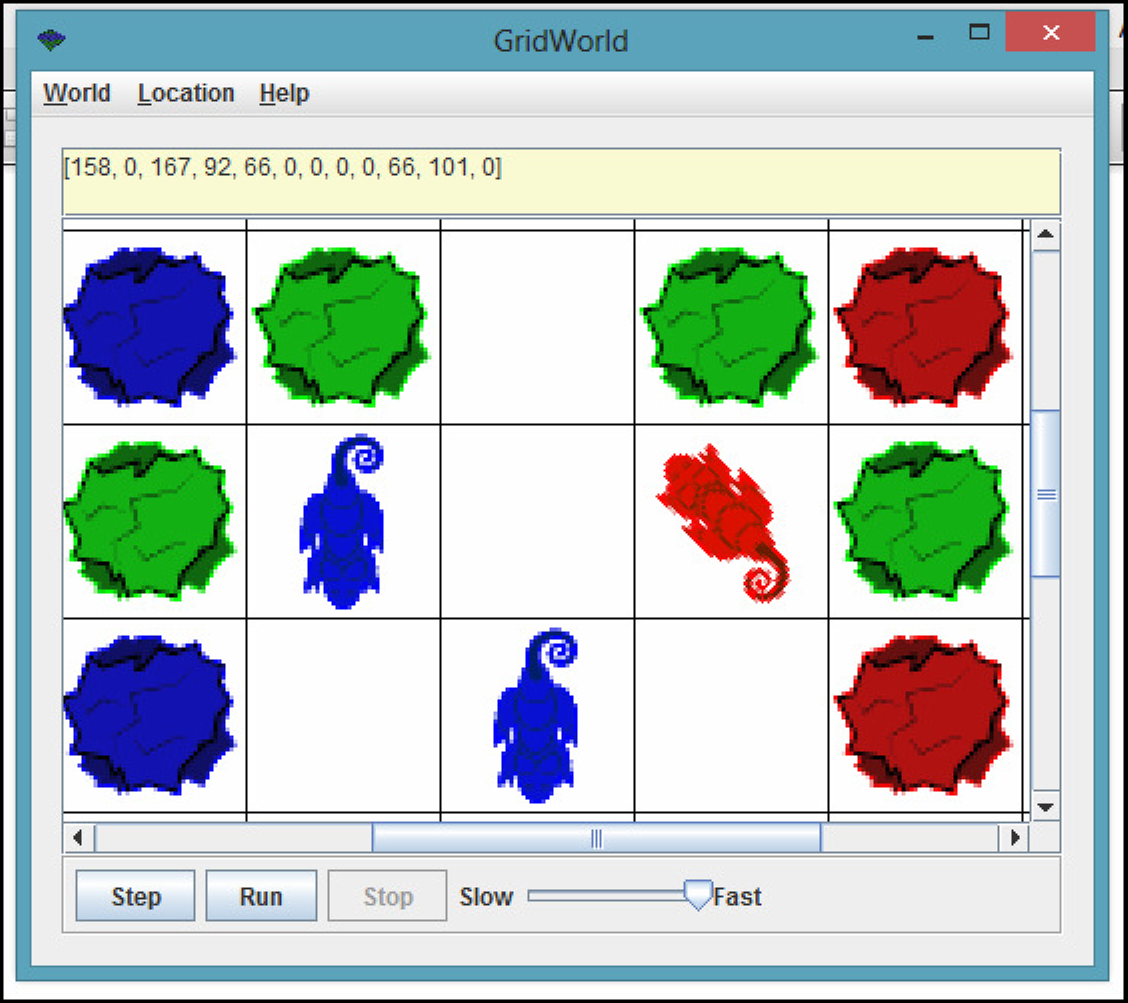


0.3333*0.3333 = 0.1111



```
[0, 233, 798, 0, 0, 0, 0, 154, 79, 0, 644, 154]
```

$$\begin{tikzcd}
 & \text{Blue} \arrow[d] & \text{Green} \arrow[dl] \\
\text{Blue} \arrow[d] & \text{Chameleon1} \arrow[dl] & \\
\text{Chameleon2} & &
\end{tikzcd}$$



[158, 0, 167, 92, 66, 0, 0, 0, 0, 66, 101, 0]