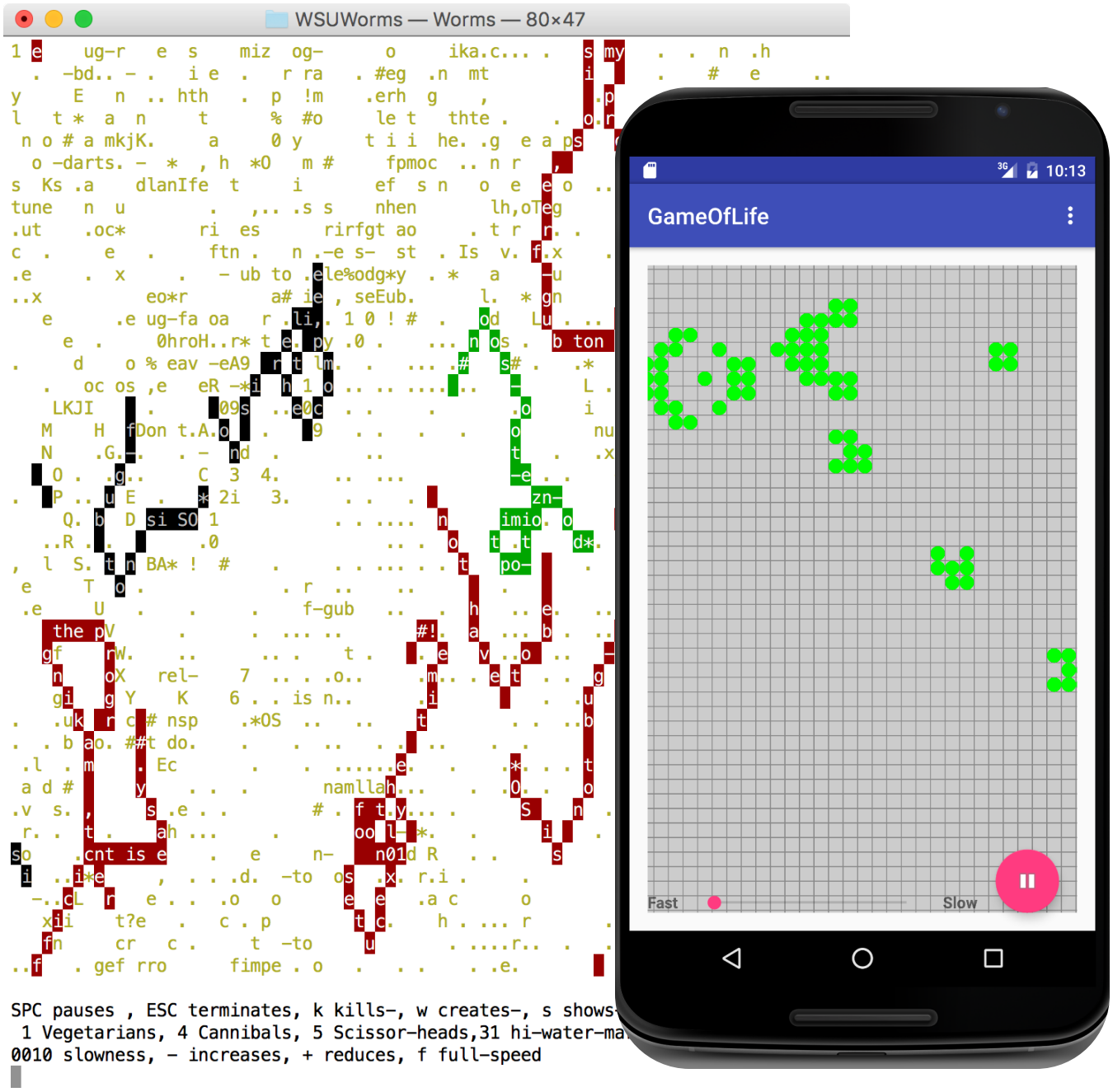
Adv. Software Eng. HW1 Summer 2016

for Wright State University

Erik M. Buck - June 1, 2016



Worms Simulation

The Worms program is a simulation of *worms* living their lives in a 2D plane called a *board*. See Figure 1: Examples of Worms On a Game Board.

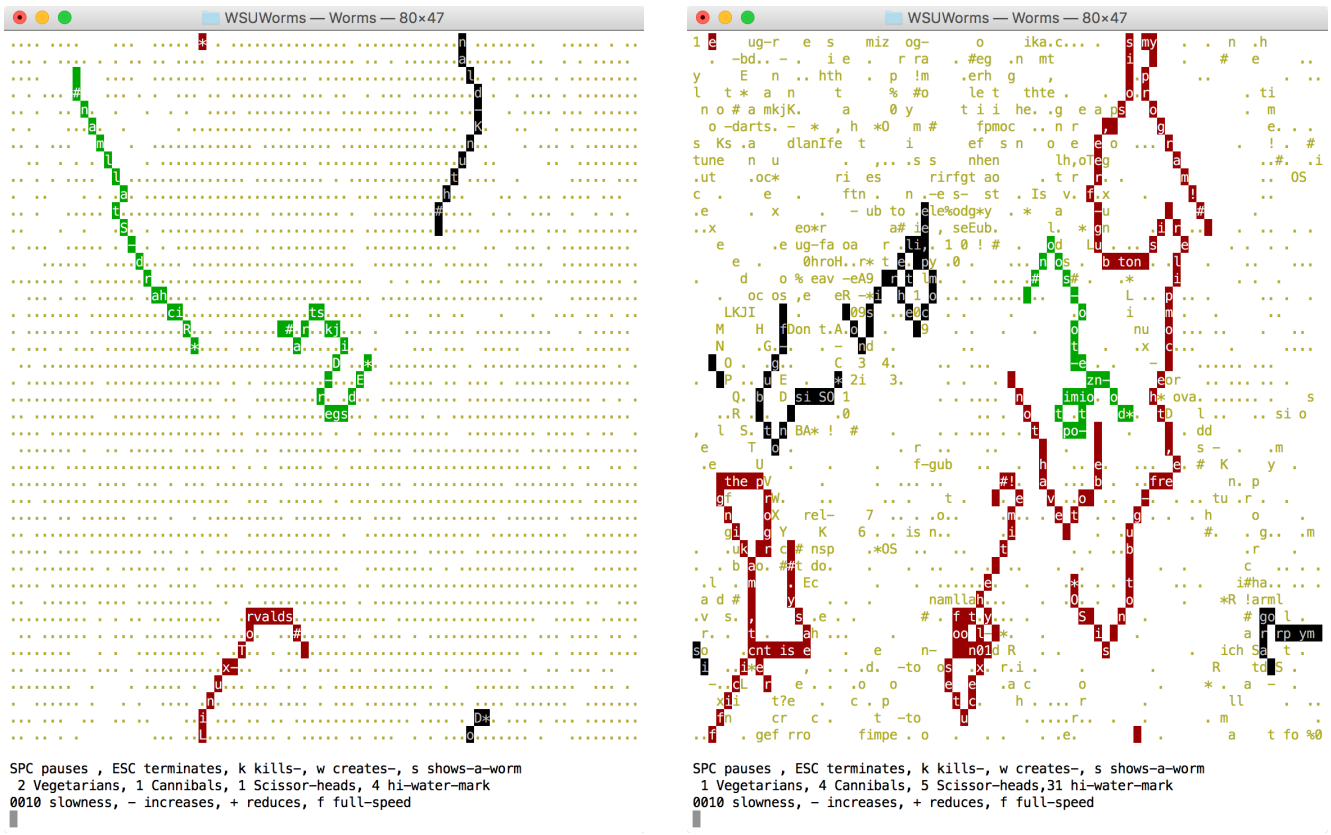


Figure 1: Examples of Worms On a Game Board

Board

The board is represented by a matrix of squares. Positions within the board are denoted by the natural number coordinates of each square in the matrix. Each square may contain zero or one element from the set *{carrot, segment from non-alive worm}* as well as zero or more worm segments of alive worms. See the Worms Section for a description of *non-alive worm, alive worm, and worm segments.* Carrots and segments of alive worms have a food value that nourishes any worm that eats them.

Worms

Three types of worm are possible: *Vegetarian, Scissor, and Cannibal*. Worms may be *alive*, *dead*, or *eaten*. Each worm is composed of one *head segment* and zero or more *non-head segments*. Each alive worm's head may move in pseudo random directions by changing its position form one position in the board to an adjacent position within the board. If a worm segment moves off an edge of the board, the worm segment "wraps" around to the opposite side of the board. Non-head segments are ordered with respect to the head and other non-head segments in the same worm. Whenever a segment moves, the immediately subsequent segment if any in the same worm moves to occupy the position just vacated by the segment that moved. In other words, the head moves around the board and non-head segments follow the preceding segment.

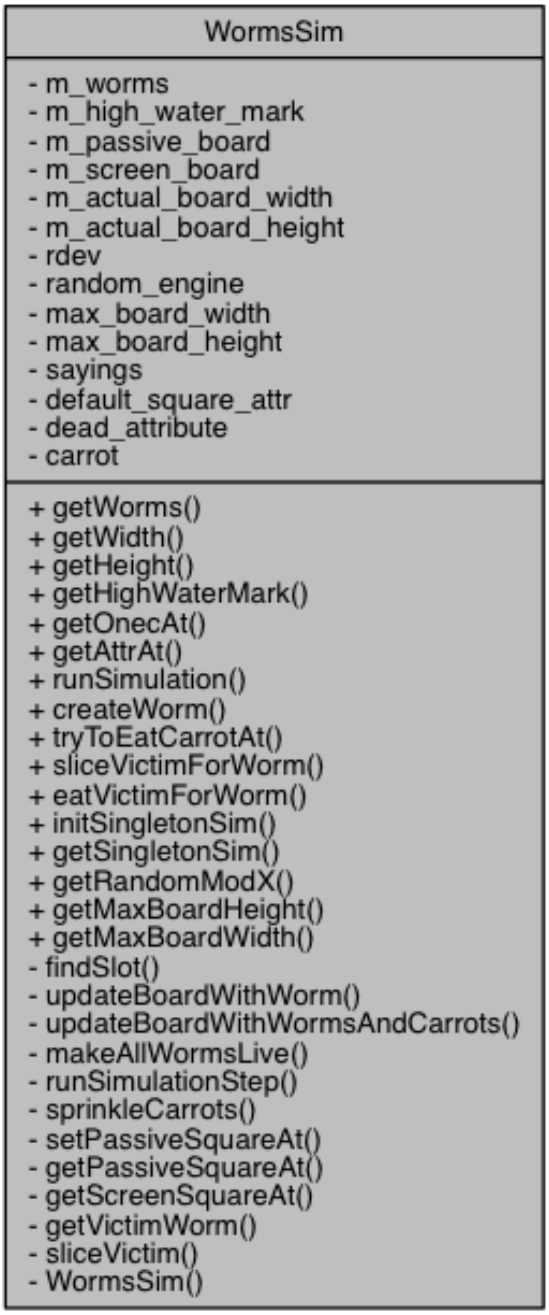
Vegetarian worms, Scissor worms, and Cannibal worms may eat any carrot at the same position in the board as the worm's head. Eaten carrots are removed from the board. Cannibal worms may also eat any entire alive worm if the alive worms is composed of one or more segments at the same position as the Cannibal worm's head. Eaten worms are removed from the board. Scissor worms may "slice" any alive worm if the alive worm is composed of one or more segments at the same position as the Scissor worm's head. When a worm is sliced, the sliced worm's segment at the same position as the Scissor worm's head is removed from the board. Any segments preceding the removed segment in the sliced worm remain part of the sliced worm. Any segments after the removed segment in the sliced worm may become part of a new worm with a head at the position of the segment immediately after the removed segment in the sliced worm.

Each alive worm has a stomach with food storage capacity proportional to the number of segments in the worm. Each time an alive worm moves, it consumes some of the food in the its stomach. Each time an alive worm eats, it increases the amount of food in the its stomach. If a worm's stomach becomes empty, the worm becomes dead.

"Proper" Object Oriented Programming (OOP) Form

The logic and data of the Worms simulation is encapsulated by three ***concrete public*** classes, *WormsSim*, *Worm*, and *CursesWormsSimUIStrategy*. An abstract class, *AbstractWormsSimUIStrategy*, is also provided as an "interface" for any class that provides display and user input for a Worms simulation.

WormSim Class



WormSim encapsulates the board, some string "sayings", and zero or more Worm instances. WormSim provides a *runSimulation()* method that when called starts or restarts a simulation with a pseudo random number of initial worms and a board containing a carrot in every square. Note: more worms than the initial number may come into existence while the simulation is running. Note: although simulation state is encapsulated by an instance of WormSim, the WormSim methods, *runSimulation(), and createWorm()*, must be called periodically to modify simulation state over time. Each created Worm instance is initialized to store a saying pseudo randomly selected from the WormSim's saying. Additionally, the WormSim methods, *tryToEatCarrotAt(), sliceVictimForWorm(), and eatVictimForWorm()*, are called by instances of Worm encapsulated by WormSim.

Design Notes:

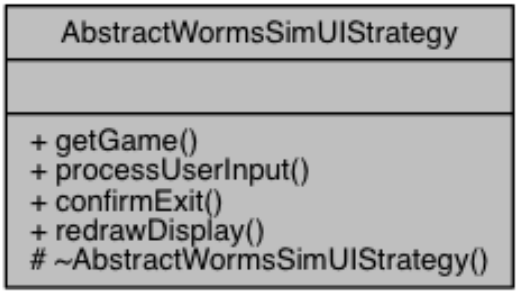
- Following the principle of separation of concerns, the WormSim class intentionally does not have any "drawing" or user input dependencies. It is anticipated that many different strategies for "drawing" a simulation board and the worms may eventually be implemented. Similarly, many different mechanisms for user input may eventually be implemented. WormSim is responsible for encapsulating board state irrespective and decoupled from drawing and user input implemented elsewhere. See Section Use of the Strategy Design Pattern for more information about the decoupling.
- Composition ("has-a") relationships are preferred over inheritance ("is-a") relationships between classes. For example, each WormsSim instance has an arbitrary

number of Worm instances and has a board. WormsSim has no “virtual” methods suitable for inheritance based polymorphic dispatch.

- WormsSim participates in the Mediator design pattern: https://en.wikipedia.org/wiki/Mediator_pattern For example, Worm instances interact with other Worm instances and the board exclusively via WomsSim.

Use of the Strategy Design Pattern

The ***abstract*** AbstractWormsSimUIStrategy class (similar to a Java or C# *Interface* or an Objective-C *Protocol*) declares the methods needed by any class that displays WormsSim state to a user and/or accepts user input. AbstractWormsSimUIStrategy collaborates as part of an implementation of the *Strategy Design Pattern*: https://en.wikipedia.org/wiki/Strategy_pattern The assumption is that many different concrete strategies for displaying WormsSim state to a users and accepting user input may exist. For example, there may be a terminal (Curses) based textual display or a 2D vector graphics display or a 3D display. Similarly, user input may come from a keyboard or over a network connection or from a data file or from script.



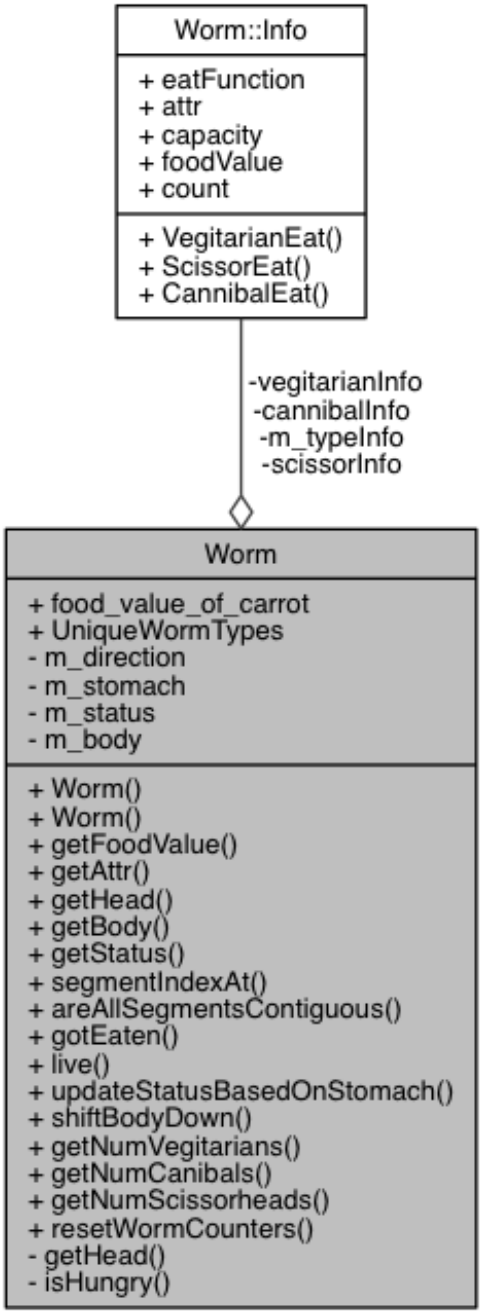
Worm Class

The Worm class encapsulates a worm that lives, eats, moves, and eventually dies or is eaten. Worm instances collaborate with a WormsSim instance to interact with the WormsSim ’s board and other Worm instances. In every case, the interaction between Worm instances is mediated (*Mediator design pattern*) by an instance of WormsSim i.e. The only way a worm instance can influence another worm instance or the board is via the methods of WormsSim.

Design Notes:

- Following the principle of separation of concerns, this class intentionally does not have any "drawing" dependencies. It is anticipated that many different strategies for "drawing" a sim worm may eventually be implemented. This class is responsible for encapsulating worm state irrespective of drawing implemented elsewhere.
- Composition ("has-a") relationships are preferred over inheritance ("is-a") relationships between classes. For example, each Worm has a reference to a

UniqueWormType and an arbitrary number of segments. An alternative design might provide an abstract "Worm" class and several distinct subclasses to encapsulate each different “kinds" of worm. However, when replicating the functionality of pmateti@wright.edu's 2013 *worms.cpp*, the only aspects of a Worm that vary by “kind” may be represented using composition (*has-a* relationships). Note: A private “inner” class accessible only within the Worm class encapsulates all differences between worm types. The kind of a worm is hidden from and irrelevant to code outside the implementation of the Worm class. It could be argued that if inheritance based polymorphism was used to enable worm kind specialization, hypothetical future Worm subclasses might be implemented as subclasses of Worm, but that is hardly the only or best approach: Future "kinds" could be accommodated through template specialization, the Visitor design pattern, the Command design pattern, or the Adaptor design pattern all of which exist to minimize coupling between code specifically by avoiding unnecessary inheritance.



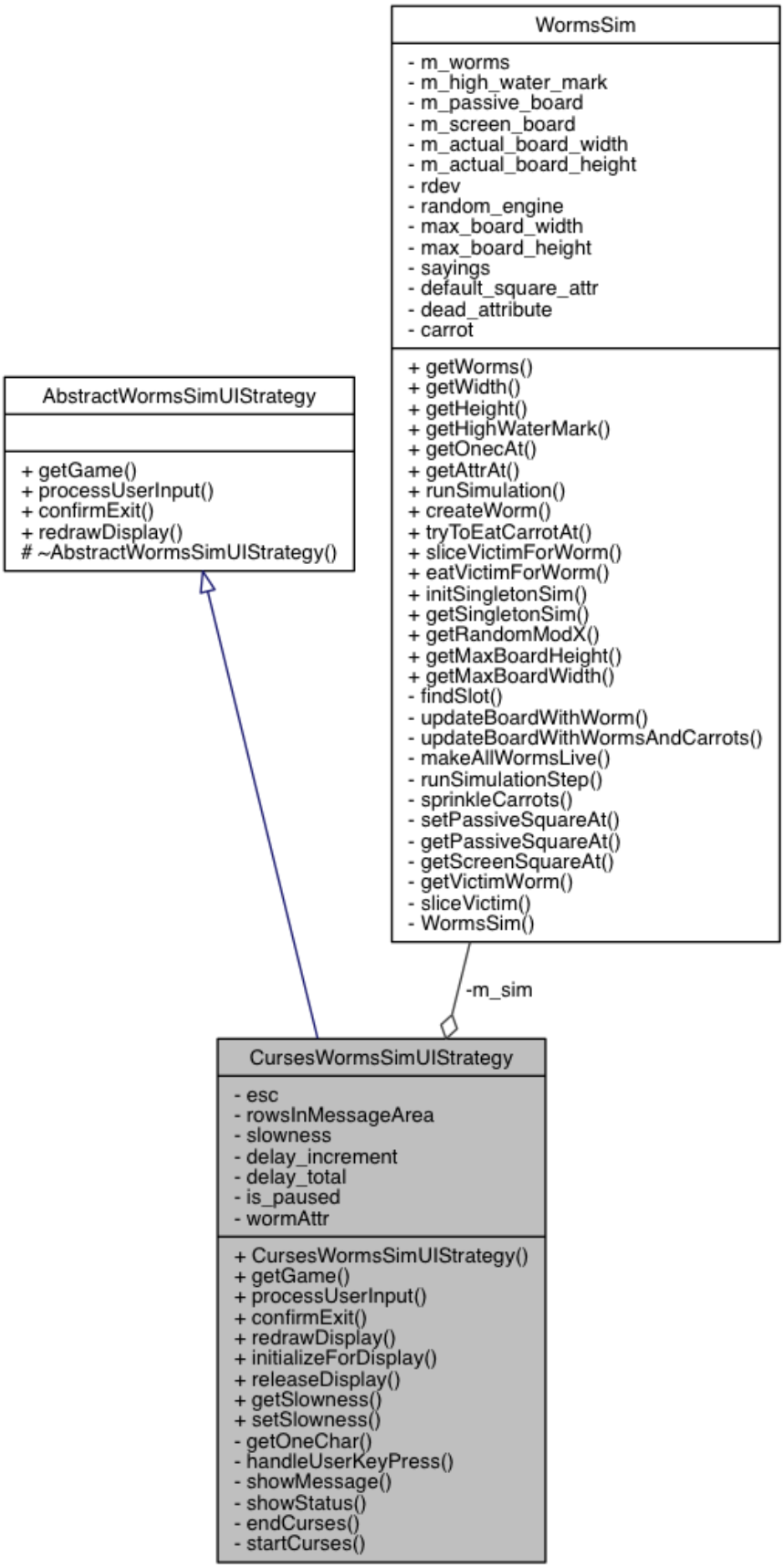
CursesWormsSimUIStrategy Class

Instances of CursesWormsSimUIStrategy provide NCurses based graphical display and user interaction of WormsSim instances.

Design Notes:

- CursesWormsSimUIStrategy implements the interface specified by the ***abstract*** AbstractWormsSimUIStrategy class and participates in the Strategy design pattern to decouple display and user input from simulation encapsulation.
- CursesWormsSimUIStrategy adheres to the practical advice from Scott Meyers, “More Effective C++: 35 New Ways to Improve Your Programs and Designs 1st Edition” Item 33, “Make non-leaf classes abstract” as elaborated to use the Template Method design

pattern in “Virtuality: C/C++ Users Journal, 19, September 2001” http://www.gotw.ca/publications/mill18.htm.



Entry And Exit Assertions for Public Methods And Class Invariants

See Worms simulation source code for assertions checking pre and post conditions within public methods and verifying class invariants. Reference the html/index.html file submitted with this document for detailed information and links to formatted source code.

Suggested Z Axis Enhancement

There are several display related challenges when depicting worm segments that may occlude each other from user view in any 2D rendering of a hypothetical 3D board. To be effectively displayed to users, the users would likely need the ability to pan, zoom, and rotate their perspective of the 3D board.

Currently, when a worm enters the board, all of the worm's segments conceptually start at the same position in the board. This roughly simulates a worm crawling onto the plane of the board from a non-existent other plane. In other words, assuming the user visible board is in a plane defined by X and Y orthogonal Axis, an imaginary Z Axis points in and out of the display perpendicular to the XY plane. It's almost as if the worm segments start aligned with the Z Axis and then one by one enter the XY plane. This imaginary behavior could be coded into reality. The simulation could be modified to let worms crawl back out of the XY plane and therefore out of user view as well.

Currently, a carrot is placed in every board square at the start of a simulation run. As a result, simulations start with copious amounts of food available to worms. If a hypothetical 3D board was also fully populated with carrots at the start of each simulation, the total amount of food available to works could be astronomical. Worms might be able to live indefinitely with so much food available.

Identified Bugs in Provided Sample Implementation

The provided pmateti@wright.edu's 2013 *worms.cpp* implementation exhibits the following bugs or unexpected behaviors.

1) On line 261 of the provided sample code, control may reach end of non-void function. This is also the root of the second bug.

2) The provided sample code, *worms.cpp*, contains an egregious "Bus error" generated at runtime from (among other places) line 384 in the *void live(WORM * wp)* function.

```
381  {
```

```
   382    SEGMENT *hp = headSeg(wp);
   383    for (SEGMENT * bp = wp->body; bp < hp; bp++) // < not <=
-> 384      bp->x = (bp + 1)->x, bp->y = (bp + 1)->y;  // crawl
   385
   386    int dir = wp->direction = (wp->direction +
nextTurn[random(16)]) % 8;
   387    hp->y += dya[dir];
```

The error is caused by unaligned or incorrect memory access. The problematic memory access occurs because *hp* minus *bp* is a large number, 114180 bytes, but the size of the entire *WORM* data structure is only 1220 bytes. The large number happens because the *hp* variable is miscomputed via the macro, *headSeg*, which evaluates to *(wp->body + wp->nsegs)*, and *nsegs* is an incorrect large value.

The *nsegs* variable value is incorrect because an error occurred earlier on line 338:

wp->nsegs = vsn - 1;

The incorrect value happened because the *vsn* argument passed to the pertinent function already stored the number 32767 and realistically probably could contain any value because *vsn* is returned implicitly by the *isAt()* function which contains line 261 - the place where an indeterminate value is implicitly returned (See Bug 1).

The root of this serious bug is easily spotted due to the compiler reporting:

```
worms.c:261:1: warning: control may reach end of non-void function
      [-Wreturn-type]
```

It is also worth noting that modern software development tools including tools I have developed as part of industry sponsored research can easily detect this kind of problem from the root cause all the way to a temporally distant error manifestation irrespective of any pre and post conditions the programmer may have seen fit to include.

By the way, my research in 2015 focussed on dynamic analysis of running software. Instruction trace data is collected at run time and analyzed off-line whenever a bug manifests. The tools my team and I developed work backward through the trace data to discover actual control flow that resulted in the error no matter how (within reason) long ago the root cause occurred. My tools work at the machine instruction level, but they have been demonstrated using compiled versions of 3 million+ line Cobol programs from the USA Social Security Administration. Our tool identified numerous bugs that were in the software in many cases from inception to after 40+ years of continuous operation. In fact, of 46 projects to date, we have discovered latent bugs in every single one.

3) When a worm is sliced and the tail portion has segments but there are no available existing slots in the list of worms, the tail portion is discarded rather than becoming a new worm.

4) Cannibals slice worms in the same way as Scissor worms and eat only one part of the sliced worm.

5) The provided sample code sets the "high water mark" of the number of worms to zero at the start of each simulation run, but "high water mark" usually refers to the highest a number has ever been as opposed to the highest it has been on a particular simulation run.

Class Discovery and Tools Used

SOLID (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion) design guidance was applied to discover the design of Worms. In addition, the design was influenced by 29 years of software development experience on projects of all sizes combined with internationally recognized expertise in software design including authorship of a seminal best selling book on the subject of design patterns: https://www.amazon.com/Cocoa-Design-Patterns-Erik-Buck/dp/0321535022

Apple's Xcode Integrated development environment and the Open Source Doxygen tools were applied. Worms source code was compiled and tested on thor.cs.wright.edu.

Game of Life

The Game of Life is a simulation employing 'cellular automaton' that was invented by Cambridge mathematician John Conway. The simulation became widely known when it was mentioned in an article published by Scientific American in 1970. It consists of a collection of cells which each, based on proximity to other cells, can live, die or multiply. Depending on the initial conditions, the cells form various patterns throughout the course of the game.

One reason the simulation is interesting is that is shows how seemingly complex self replicating "machines" can be composed of simple "machines". Composite behaviors of life can emerge for startlingly simple individual cell behaviors. This simulation was a precursor for related scientific explanations of fish schooling, bird flocking, lichen growing, and disease propagation. See https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

The Game of Life simulation consists of a 2D *board* that is for practical purposes infinite in size. The board uses signed 32-bit integer {x, y} coordinates to identify each position on the board that can host a cell. There are a more than 1.6 times 10^{19} possible cell positions. As a practical matter, if a user observed 1600 cell positions per second, it would take the user 317 million years to observe every possible cell position. Note: A standard Java IEEE-754 32-bit floating point coordinate system is used for actual display, and mild gradually increasing graphical distortion of displayed cell positions is likely after the user scrolls the display for approximately 10 years. The distortion is caused by insufficient precision in the mantissa portion of 32-bit floating point values. If Java had used IEEE-754 64-bit double precision floating format, the distortion would not occur within 317 million years of scrolling. See Figure 2: Game of Life Screen Shots.

Zero or more *cells* that live out their lives on the board and potentially reproduce to create new cells in adjacent positions on the board. The simulation is typically started with an initial configuration of cells populating the board. Once the simulation starts, the following rules are applied at each simulated moment in time:

1 Any live cell with fewer than two adjacent live neighbor cells dies.

2 Any live cell with two or three adjacent live neighbor cells lives on to the next simulated moment in time.

3 Any live cell with more than three adjacent live neighbor cells dies.

4 Any unpopulated board position with exactly three adjacent live neighbors becomes a live cell, as if by reproduction.

Design and Implement in Java for Android

The Game of Life Java Android application enables user selection of simulation starting conditions, user control of the real-time interval between simulated moments in time, pause and resume of simulations, and touch gesture based pan (scroll) around the simulated board.

The simulation models simultaneous actions of all living cells by applying rules to every living cell during every simulated moment in time. Each cell reacts to identical simulation state as every other cell during the same simulated moment in time.

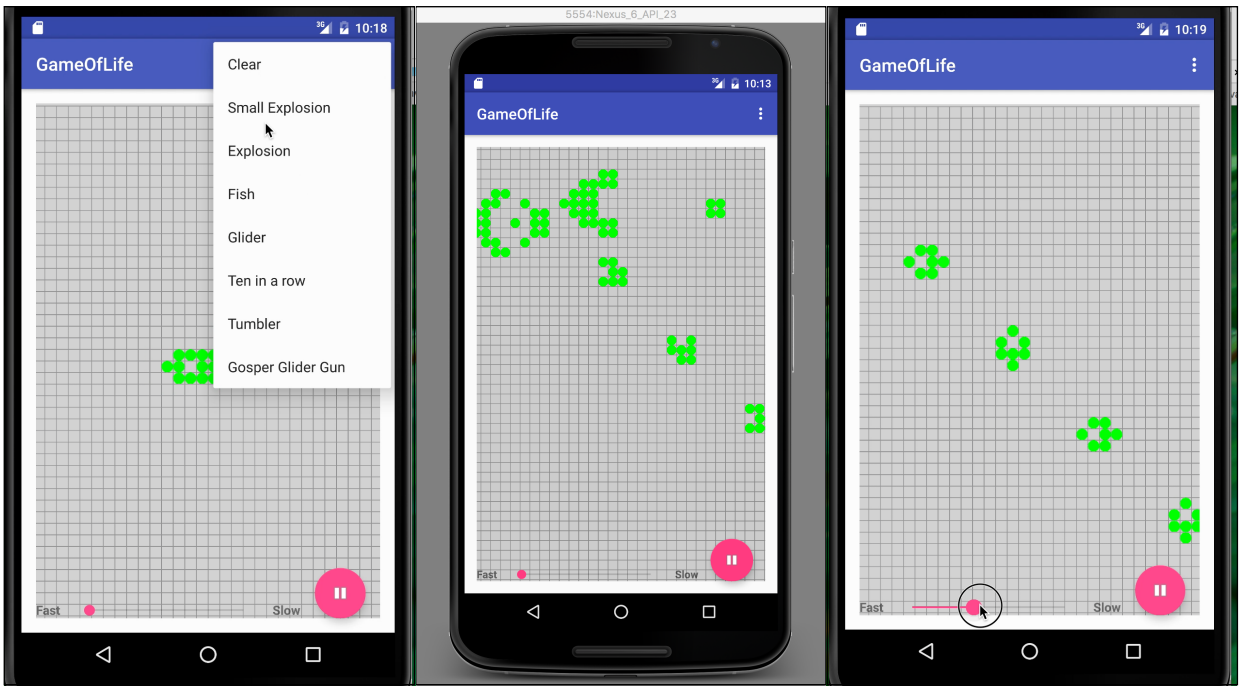


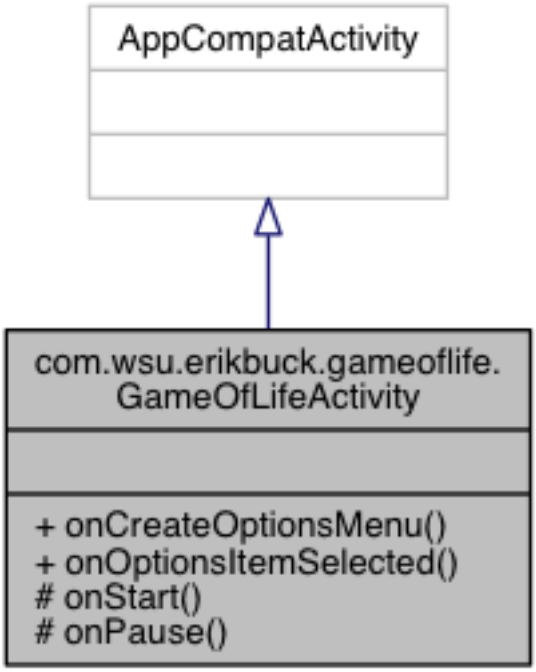
Figure 2: Game of Life Screen Shots

The Game of Life Java Android application was developed using Google's free Android Studio integrated development environment. Android studio automatically generates several Java source code files for every developed application. The automatically generates files are considered non-user-editable implementation details of the integrated development environment and Java, and are therefor not described here.

The developed portion of the Game of Life implementation consists of the following Java classes: GameOfLifeActivity, GameOfLifeModel, GameOfLifeView, and PanCapableView.

GameOfLifeActivity Class

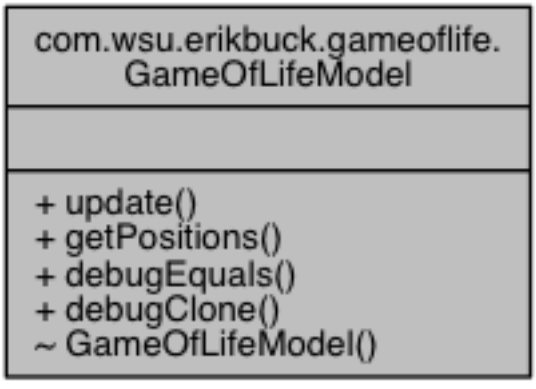
Every graphical Android Java application has at least one instance of the Android.Activity class, and in the case of the Game of Life application, that one instance is actually an instance of the GameOfLifeActivity subclass of Android.AppCompatActivity. The GameOfLifeActivity instance creates and interacts with a GameOfLifeModel instance to implement the simulation and one or more Android.View instances to provide a user interface for the simulation. Android Activities make extensive use of the Template Method design pattern: See https://en.wikipedia.org/wiki/Template_method_pattern All public methods of GameOfLifeActivity are overrides of Android.Activity Template Methods.



GameOfLifeModel Class

Each instance of the GameOfLifeModel class encapsulates an arbitrary number of cells which apply the logic (rules) for the Game of Life documented at http://www.bitstorm.org/gameoflife/. Cells conceptually exist in a regular 2D grid using signed integer coordinates to identify each grid position that may be occupied by a cell. At all times, there is at most one cell at each grid position.

This design does not employ concurrency such as distributed processing or multiple threads within the implementation of the GameOfLifeModel class. There are several reasons that concurrency is unlikely to produce any performance improvement when executing the simulation. The primary reason is that all cells must have access to the data structure used to store cells, and either the data structure must be copied into each thread (pass by value) or more than one thread must access and mutate the same data structure. To prevent data corruption caused by simultaneous mutation of the same data structure, mutual exclusion locks are needed. If the data structure is copied, the time required to make each copy will completely dwarf an time savings achievable through



concurrent execution. If mutual exclusion locks are used, the intended concurrent operations will in fact become sequential due to the mutual exclusion preventing simultaneous actions, and no time savings will be realized. A secondary reason is that at least one O(n) operation (where n is the number of cells) needs to be performed for each simulated moment in time regardless of any other concurrency implemented, and performing cell logic during the O(n) operation does not increase the computational complexity (calculation time) meaningfully. In summary, concurrency cannot significantly speed the actual performance of the simulation, and including cell simulation logic in an existing non-concurrent algorithm does not significantly reduce actual performance.

Instead of actual concurrency, the rules for evaluating every cell's logic are applied sequentially, but the logic for evert cell is evaluated once per simulated moment in time. In other words, upon each update of the game, all cells evaluate the same environment prior to any cells modifying the environment. As a result, the order in which cells evaluate the environment is immaterial.

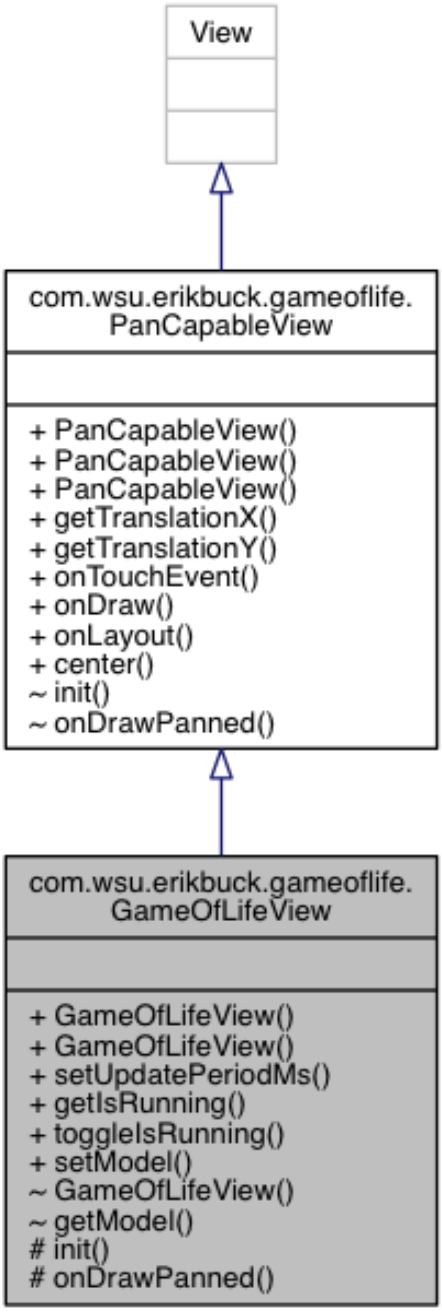
This implementation has no dependencies on the way a Game of Life model may or may not be displayed to users. After construction, call the update() method periodically to update the game state. The frequency of calls to update() does not affect game logic (rules).

PanCapableView Class

The PanCapableView class is inspired by sample code provided at http://vivin.net/2011/12/04/implementing-pinch-zoom-and-pandrag-in-an-android-view-on-the-canvas/8/ Author: Vivin Paliath and modified by Erik M. Buck. The class implements pan gesture handling. It has been greatly simplified and modernized from Vivin Paliath's version and now supports "inflation" from an XML layout as well as automatic handling of view layout and re-layout by Android.

GameOfLifeView Class

Each instance of the GameOfLifeView class encapsulates and instance of GameOfLifeModel and displays the current state of the encapsulated GameOfLifeModel each time GameOfLifeView's onDrawPanned() method is called. Note: onDrawPanned() is called automatically by the superclass, PanCapableView, and should not be called at any other time from any other place. The implementation of onDrawPanned() relies upon consistent appropriate external state provided by PanCapableView prior to calling onDrawPanned().



Each instance of GameOfLifeView periodically calls its encapsulated GameOfLifeModel's update() method. The frequency of the calls is determined by the update period in milliseconds specified using the setUpdatePeriodMs() method and whether or not the GameOfLifeView is "running". If setUpdatePeriodMs() is not called, a reasonable default update period is used.

Entry And Exit Assertions for Public Methods And Class Invariants

See GameOfLife simulation source code for assertions checking pre and post conditions within public methods and verifying class invariants. Reference the html/index.html file submitted with this document for detailed information and links to formatted source code.

Game of Life Specification

The following specifications are suitable for translation into executable Scheme code.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ADT Map algebraic specs
;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CANONICAL CONSTRUCTORS
;; pos : Pos(x, y) where x and y are signed integers
;; state : State
;; cell : Cell x Pos x State -> Cell

;; CONSTRUCTOR AXIOMS
;; isAbsent(state) = true

;; OBSERVERS
;; getAdjacent : Pos -> {Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell} where {}
denotes a set
;; filterCount : F x {Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell} -> n
       where n is natural number
       where F is a function of the form F x Cell -> Boolean
```

```
      and n is the number of items from {Cell, Cell, Cell, Cell, Cell, Cell, Cell,
Cell} for which F returns true.
;; isAbsent : State -> Boolean
;; isSpawning : State -> Boolean
;; isAlive : State -> Boolean
;; getPosition : Cell -> Pos
;; getState : Cell -> State
;; updateState : Cell -> Cell
;; becomeAlive : Cell -> Cell

;; OBSERVER AXIOMS
;; getAdjacent : Pos -> {Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell}
;;     For all p in Pos and all s1, s2, s3, s4, s5, s6, s7, s8 in State
;;          adjacent(p) = { Cell(Pos(p.x - 1; p.y - 1), s1),
                            Cell(Pos(p.x, p.y - 1), s2),
                            Cell(Pos(p.x + 1, p.y -1), s3),
                            Cell(Pos(p.x - 1, p.y), s4),
                            Cell(Pos(p.x + 1, p.y), s5),
                            Cell(Pos(p.x - 1, p.y + 1), s6),
                            Cell(Pos(p.x, p.y + 1), s7),
                            Cell(Pos(p.x + 1, p.y + 1), s8)
                          }
;; isAbsent :  State -> Boolean
;;    For a s in State
;;        if isAlive(s) = false and isSpawning(s) = false then
              isAbsent(s) = true
;;        else isAbsent(s) = false
;; isSpawning :  State -> Boolean
;;    For a s in State
;;        if isAlive(s) = false and isAbsent(s) = false then
              isSpawning(s) = true
;;        else isSpawning(s) = false
;; isAlive :  State -> Boolean
;;    For a s in State
;;        if isSpawning(s) = false and isAbsent(s) = false then
              isAlive(s) = true
;;        else isAlive(s) = false
;; getPosition : Cell -> Pos
;;    For all s in State and all x in signed integer and all y in signed integer
;;        getPosition(Cell(Pos(x, y), s) = Pos(x, y)
;; getState : Cell -> State
;;    For all p in Pos and s in State
;;        getState(Cell(p, s)) = s
;; updateState : Cell -> Cell
;;    For all c in Cell and sAb for which isAbsent(sAb) = true and sA for which
isAlive(sA) = true and all sS for which isSpawning(sS) = true
```

```
;;        if 2 > filterCount(isAlive, getAdjacent(getPos(c)) then
              updateState(c) = Cell(getPosition(c), sAb)
;;        else if 3 < filterCount(isAlive, getAdjacent(getPos(c)) then
              updateState(c) = Cell(getPosition(c), sAb)
;;        else if isAbsent(getState(c)) and 3 = filterCount(isAlive,
getAdjacent(getPos(c)) then
              updateState(c) = Cell(getPosition(c), sS)
;;        else updateState(c) = c
;; becomeAlive : Cell -> Cell
;;    For all c in Cell and sA where isAlive(sA) = true
;;        if isSpawning(getState(c)) = true then
;;            becomeAlive(c) = Cell(getPosition(c), sA)
;;        else becomeAlive(c) = c
```

How to Design a Test Plan for Concurrency and Infinite Board

Design by Contract implemented as pre and post condition assertion can be shown to be equivalent to the rigorous specification in the preceding section. Any Test Plan to verify concurrency and)as far as practical) infinite board will include demonstration that no pre or post condition assertions fail combined with analysis to show that all possible control flow patterns through the implementation code executed during the demonstration.

A tool like the open source KLEE http://llvm.org/pubs/2008-12-OSDI-KLEE.html KLEE is capable of automatically generating tests that achieve high coverage on a diverse set of complex and environmentally-intensive programs. KLEE is capable of testing all possible control flow sequences in tiny programs like Game of Life.

Clean Build with Java 8

Please see Android Studio 2.0 project delivered with this document. Java Lint reports "No suspicious code found". Building the project source code produces the following output:

```
Information:Gradle tasks
[clean, :app:generateDebugSources, :app:prepareDebugUnitTestDependencies, :app:mockab
leAndroidJar, :app:generateDebugAndroidTestSources]
:clean UP-TO-DATE
:app:clean
:app:checkDebugManifest
:app:prepareComAndroidSupportAnimatedVectorDrawable2330Library
:app:prepareComAndroidSupportAppcompatV72330Library
:app:prepareComAndroidSupportDesign2330Library
:app:prepareComAndroidSupportRecyclerviewV72330Library
```

```
:app:prepareComAndroidSupportSupportV42330Library
:app:prepareComAndroidSupportSupportVectorDrawable2330Library
:app:prepareDebugDependencies
:app:compileDebugAidl
:app:compileDebugRenderscript
:app:generateDebugBuildConfig
:app:mergeDebugAssets
:app:generateDebugResources
:app:mergeDebugResources
:app:processDebugManifest
:app:processDebugResources
:app:generateDebugSources
:app:prepareDebugUnitTestDependencies
:app:mockableAndroidJar
:app:prepareDebugAndroidTestDependencies
:app:compileDebugAndroidTestAidl
:app:processDebugAndroidTestManifest
:app:compileDebugAndroidTestRenderscript
:app:generateDebugAndroidTestBuildConfig
:app:mergeDebugAndroidTestAssets
:app:generateDebugAndroidTestResources
:app:mergeDebugAndroidTestResources
:app:processDebugAndroidTestResources
:app:generateDebugAndroidTestSources
Information:BUILD SUCCESSFUL
Information:Total time: 5.478 secs
Information:0 errors
Information:0 warnings
Information:See complete output in console

Information:Gradle tasks
[:app:generateDebugSources, :app:prepareDebugUnitTestDependencies, :app:mockableAndro
idJar, :app:generateDebugAndroidTestSources, :app:compileDebugSources, :app:compileDe
bugUnitTestSources, :app:compileDebugAndroidTestSources]
:app:checkDebugManifest
:app:prepareDebugDependencies
:app:prepareDebugUnitTestDependencies
:app:prepareDebugAndroidTestDependencies
:app:compileDebugJavaWithJavac
:app:compileDebugSources
:app:compileDebugUnitTestJavaWithJavac
:app:compileDebugUnitTestSources
:app:compileDebugAndroidTestJavaWithJavac
:app:compileDebugAndroidTestSources
Information:BUILD SUCCESSFUL
Information:Total time: 1.936 secs
```

```
Information:0 errors
Information:0 warnings
Information:See complete output in console
```

Class Discovery and Tools Used

The GameOfLifeActivity class and GameOfLifeView class were both essentially mandated by the Android Studio 2.0 integrated development environment. No Android Studio 2.0 Java project can be built without a custom Android.Activity subclass and at least one custom Android.View subclass.

The PanCapableView class was created based on the principle of separation of concerns. Panning based on user touch input is not a feature unique to the GameOfLife application and therefore was factored into its own class.

The GameOfLifeModel class contains all of the non-user-interface related logic of the Game of Life including a concrete implementation of the rigorous specification provided in this document.

The entire Game of Life application follows the Model View Controller design pattern where GameOfLifeModel is the Model, GameOfLifeView is the View, and GameOfLifeActivity is the Controller.

Android Studio 2.0 including a Java 8 complete and Java Lint were used along with the Open Source Doyxgen tool.

Note: To enable assertions, make sure the application is compiled with DEBUG enabled. In Android Studio, use the Build->Edit Build Types... menu. Select the "Debug" build type and make sure the "debuggable" option is set to true. Assertions make my application run a 1/20th the speed it runs without the assertions enabled.

Development Journal

Version Control Log for Worms:

```
commit 83b006db10598ea8f6585ebd47e2806e2a290667
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 24 15:08:33 2016 -0400

    Replaced unneeded use of pointer with reference.

commit 7125d05eef23be48c1aefaa95f854c9c501d57cd
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Sat May 21 14:58:41 2016 -0400

    Improved implementation comments.

commit ffdb9fdde82d6d59d3e4992b66e57e8fcbc39f5c
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Sat May 21 14:58:23 2016 -0400

    Improved class documentation.

commit eb1660fd3110e23ce7704e6ecccc3b187677613d
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Sat May 21 14:57:48 2016 -0400

    Improved copyright notice

commit 1ea6205bd3df3f8ff111ac1dd3199621b6c86d0b
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:32:42 2016 -0400

    Modified copyright assertion.

commit 788a2e3ebe1271288d6099fb13e8cb1728f79c58
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:28:43 2016 -0400

    Added posX and posY parameters to Worm constructor.

commit 5490a87dd6a0d36b8046f10963718be1664c5bf1
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:27:58 2016 -0400

    Added posX and posY parameters to Worm constructor.
```

```
commit 915a37fa2a6a8a6dcec6429fbc7bdb507a44f595
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:27:14 2016 -0400

    Moved random aspects of of Worm instance generation into WormSim::createWorm()
function.

commit 8d59fca67561854c7be00b583d226e04a94e848a
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:26:02 2016 -0400

    Moved selection of worm kind and saying into Worm constructor.

commit b8d1714b545ce68a14957cc909124636c16c65b8
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:07:06 2016 -0400

    Renamed UIStrategy related classes.

commit ef191d64e528b1da0af9a96289a6182c55f591d4
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Fri May 20 20:01:46 2016 -0400

    Renamed classes and slightly refactored, made use of Strategy design pattern
explicit, and added Oxygen formatted comments.

commit cc680698dfee6bb122543daacee42eaa1a758ed2
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 13:46:56 2016 -0400

    Added #include <algorithm>

commit 66acafd8319d29dfabc28d72210031824af6d8bd
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 13:46:32 2016 -0400

    Changed return type of getHighWaterMark().

commit e05114c526c0a5503f0eb62a6e06116b31180a62
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 13:46:03 2016 -0400

    Added include and corrected formatting type in snprintf().

commit 6c4039e2056dc6e45887b3fd8e5dfa5c69b9f01f
```

```
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 12:17:00 2016 -0400

    Removed last Curses dependencies from Worm and WormsGame classes.

commit 3ffaa9520f36591cf317e779dc2f246355e721d1
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 11:37:37 2016 -0400

    Reordered files.

commit 4f9a38ca84a826054ed3eaf82d341a28922748c6
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 11:36:55 2016 -0400

    Moved implementation details from .h file to .cpp file.

commit b601c7bc17b2c02b541e0fb32f782008e82916f0
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 11:30:11 2016 -0400

    Moved implementation details from .h files to .cpp files.

commit 356c4f0e92af089ddc6962ec7f7f67504630e3d7
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 11:06:44 2016 -0400

    Moved Curses specific code into CursesDrawWormsGameDecorator.

commit 09d8af02e90c57cde2bd7a62b53ef8d3b092ed84
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 18 10:32:10 2016 -0400

    Standardized method naming convention and moved some portions of implementation
into Worm.cpp.

commit 1f4f6ea88cab4a32775adaa14cdcae1e18daa27d
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 17 19:55:51 2016 -0400

    Improves display colors and status display.

commit 678e6d9f2c987104cc1c0a95365d560b8a3aea8c
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 17 19:26:03 2016 -0400
```

```
    First refactor.

commit 7f72f71f1587c2b006286d988ebe07caa1401b9d
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 16 15:45:13 2016 -0400

    Made minimum changes needed to compile on OS X with ncurses installed.

commit 09a3a08e7740c3d15bd2b62f9278580a27001ef7
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 16 14:58:21 2016 -0400

    Initial version from http://cecs.wright.edu/~pmateti/Courses/7140/Lectures/
Examples/Worms/worms-one-file.cpp.html

commit 2183661b9cea3a16903c04fef8aaf92608a573c8
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 16 14:57:35 2016 -0400

    Initial Commit
```

Version Control Log for GameOfLife:

```
commit 2fdb5d8a1e1c84c833cd867f5707ece7ea867895
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Sat Jun 18 17:30:18 2016 -0400

    Implemented suggestion generated by Java Lint

commit b5ffc3e075fb079e95f6f4fc62c00a712e8c2f14
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed Jun 8 12:14:15 2016 -0400

    Update README.md

commit 891aad191f8a415b4535dbd4c1c113047ed412ab
Merge: 35bd665 78916c9
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed Jun 8 12:07:25 2016 -0400

    Merge branch 'master' of https://github.com/erikbuck/GameOfLife-for-Android

commit 35bd6657eaef398724714ea89df9208e7b18fa40
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed Jun 8 12:07:07 2016 -0400
```

```
    Added files

commit 78916c9bd877a0bd4c359b9fb1aac45d82c4d6dc
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed Jun 8 12:05:01 2016 -0400

    Create README.md

commit 81e10720980ca08d534f5454c87322a39b40fd63
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Thu May 26 19:04:11 2016 -0400

    Rearranged methodd declaration order  and refined pre and post conditions for
methods.

commit 06c73d3c13f38f82f06f9b58fbd586a10d8977e2
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 23:30:58 2016 -0400

    Fully implemented pre and post conditions.

commit f13e21fd7e941a212e02ae43fbf0c703e8bd5400
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 21:51:33 2016 -0400

    Added notation about enabling assertions.

commit cd7caffcbdb5d04acd84f33ce28a234d08907e16
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 21:45:26 2016 -0400

    Standardized top of class block comments for JavaDoc.

commit 1cc7339bbd67607c9b67b3c26b550c3886bacdc4
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 21:42:27 2016 -0400

    After much searching and hand wringing, I have confirmed that Google's Dalvik
virtual machine for Android does not support Java's built-in assert. There is no
reliable way to persuade Android to do anything useful with built-in  assert. Dalvik
just does not built to support it.

    I have replaced all uses of built-in assert with if(BuildConfig.DEBUG)
assertTrue().
```

The test for if(BuildConfig.DEBUG) avoids evaluation of the assertion's conditional when not compiling for DEBUG. This is necessary because running with assertions enabled results in approximately 1/20 application performance compared to without assertions enabled.

```
commit 89512647d436eff6708b3ab6a088312f074909cb
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 18:07:37 2016 -0400
```

Made Java lint recommended changes and noted where Java lint assures the language guarantees the assertions cannot fail.

```
commit 23b26d470d5576aeae0e76a426edcf399c731a47
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 18:01:16 2016 -0400
```

Added assertions and converted built-in assert to Assert.assertTru() because AndroidSTudio and Android do not use the built-in assert. This is likely part of teh contention betweenGoogle and Oracle regarding Java languagespecifications and conformance.

```
commit c3aa22dd9207e1ecb072eb994eb8f881f07a3185
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 16:54:48 2016 -0400
```

Added copious JavaDoc comments and added some assertions for pre and post conditions.

```
commit 07068fd7ce68d116aca8de8d93bda5b03171fda6
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Wed May 25 14:06:33 2016 -0400
```

Made changes suggested by Java lint.

```
commit 934284fd4d98b17771cd9c559a64af7328a1a540
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 24 14:20:04 2016 -0400
```

Added strings used as menu item labels.

```
commit 6f0b30517ebb6a94025e4a69b97de685ce9bc0cc
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 24 14:17:36 2016 -0400
```

Converted appropriate public and protected element to private. Reformatted code.

```
commit 5e0e3656aa36d353de4ce48a561975004d150a65
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 24 14:16:52 2016 -0400

    Added layout support and SeekBar with labels.

commit b309c89a93bc8c6b1e2e1861e30f853796cbe7b7
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Tue May 24 13:36:15 2016 -0400

    Added support for SeekBar based update period selection.

commit cb281aec332a39b7717e63ba5c713a7c0cc98b46
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 23 22:07:41 2016 -0400

    Added support for the standard set of starting conditionsselectable through a
standard "settings" menu.

commit a2ea934c81825e583cd199e55ac9f222e23780b2
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 23 18:57:45 2016 -0400

    Initial  version that appears to work.

commit 20f9331310376573c883a148b5f3e91a8234fd83
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 23 14:05:35 2016 -0400

    Initial version.

commit ff6c7c586c32f0db28f522781679f351f6f185f8
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 23 14:03:50 2016 -0400

    Integrated sample code for zoom and pan from http://vivin.net/2011/12/04/
implementing-pinch-zoom-and-pandrag-in-an-android-view-on-the-canvas/8/

commit 2a58c3bbe10c2a0cdd4ad29046dde98d4629d3de
Author: Erik M. Buck <erik.buck@sbcglobal.net>
Date:   Mon May 23 14:03:16 2016 -0400

    Added GameOfLifeModel and implemented drawing the game grid.

commit a9691dff93c88bda90625204ec2ea3e024d81e2f
Author: Erik M. Buck <erik.buck@sbcglobal.net>
```

```
Date:   Mon May 23 00:04:11 2016 -0400

    Implemented pan gesture.
```