ArtIn: Survival

September 8th 2013

Version 1.1

|  |  |
| --- | --- |
| **Name** | **Title** |
| Roger Baumbach II | Lead Designer |
| Tyler Helwig | Lead Programmer |

# Vision and Scope

## Problem Statement

### Background

ArtIn: Survival creates and simulates an artificial intelligence (AI) living in a randomly generated world. The program will contain the AI and a method by which a world may be generated. The program will allow state saving of a simulation for continuation at a later time.

The initial program will be written to simulate multiple animals attempting to survive in a randomized landscape. In a survival situation, the intelligence (artificial or otherwise) must find resources and conserve energy. Sufficient modeling software is not easily obtainable to re-create this environment in a manner that is easily changed.

This program can be used for various situations, such as to simulate bacteria living in an aqueous environment or a man attempting to survive in a desert. The simulated creature will be written with the average human’s survival needs. For example, an average human’s death typically occurs after 10 to 14 days [5] without food. Other things show any range from a few days to a few months. We will be using the rule of three for any of our situation. The rule of three states that a human can survive 3 minutes without air, 3 days without water, and 3 weeks without food. [6] These are the extremes though and the AI will perish if it hits one of the limits. The program will be written to be easily adaptable to any similar situation and with different environmental factors that may play a part.

### Stakeholders

* Roger Baumbach CEO, lead AI programmer, and technical writer.
* Tyler Helwig Lead interface programmer, QA, and PR.

### Users

* Modeling scientists
* Programmers that wish to have a self-building populace
* People interested in simulation

### Risks

As the world generation is theoretically infinite, we run the risk of running out of resources. Checks will have to be put in place to avoid this situation.

Many variables affect the behavior of an animal, so we may not be able to accurately reproduce behavior in the given timeframe.

### Assumptions

We assume that a user has a computer with Java installed. This program will work on all java-compatible systems.

## Vision of the solution

### Vision statement

The product will provide an easy to use, accurate model of some of the most common simulations desired. It provides modeling scientists with a way of simulating environments without needing to know how to code the environment themselves. The market for scientific and casual simulation has been growing, but there is no product that is both highly adaptable and easy to use. The core artificial intelligence algorithm is entirely proprietary and difficult to duplicate, giving us a boost over potential competitors. The product can be easily adapted into a video game, providing us several avenues of revenue generation. These benefits give us strong justifications to develop this product.

### List of features

* It builds a theoretically infinite world on-demand.
* It simulates any number of artificial intelligences.
* It can create a range of environment types.
* It can be run at various speeds.

### Features not supported

* It does not support direct intervention in the world.
* The speed of simulation is limited to the power of the machine it is running on.

# References

[1] Oracle. (2013). *Java™ Platform, Standard Edition 7 API Specification.* [Online]. Available: <http://docs.oracle.com/javase/7/docs/api/>

[2] Stackoverflow. (2013). *A lot of lag when using g2d.drawImage - Java*. [Online]. Available: <http://stackoverflow.com/questions/17396331/a-lot-of-lag-when-using-g2d-drawimage-java>

[3] Wikipedia. (1 December 2013). *Artificial Intelligence.* [Online]. Available: <http://en.wikipedia.org/wiki/Artificial_intelligence>

[4] Stackoverflow. (2013). *Hide/Show JFrame using Jbutton on JPanel.* [Online]. Available: <http://stackoverflow.com/questions/15051247/hide-show-jframe-using-jbutton-on-jpanel>

[5] Scientific American. (8 November 2004). *How long can a person survive without food?.* [Online]. Available: <http://www.scientificamerican.com/article.cfm?id=how-long-can-a-person-sur>

[6] Ruleof3Survival.com. (2013). *Survival Skills.* [Online]. Available: <http://www.ruleof3survival.com/>

[7] Wikipedia. (2 November 2013). *Variable (computer science).* [Online]. Available: <http://en.wikipedia.org/wiki/Variable_(computer_science)>

[8] Wikipedia. (3 December 2013). *Java (programming language).* [Online]. Available: <http://en.wikipedia.org/wiki/Java_(programming_language)>

# Definitions

|  |  |
| --- | --- |
| Term | Definition |
| Artificial Intelligence (AI) | Artificial intelligence (AI) is technology and a branch of computer science that studies and develops intelligent machines and software. Major AI researchers and textbooks define the field as "the study and design of intelligent agents", where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success.[3] |
| World | Randomly generated environment that the AI will interact with. |
| Tile | A simulated 1 meter by 1 meter block that contains one type of environmental feature. |
| Chunk | A 30x30 two-dimensional tile array. |
| Window | A User interface that allows a user to interact with the program. |
| Variable | In computer programming, a variable is a storage location and an associated symbolic name (an identifier) which contains some known or unknown quantity or information, a value. [7] |
| System | The programs environment in which it operates. |
| Animal | Used to refer to our Artificial Intelligence. |
| Java | Java is a computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. [8] |
| Tick | An incremental step in the program. Each tick will fire a set of predefined methods that will intern force the actions of each entity then repaint the window or frame with the results and movements. Each tick takes several real world seconds and in turn is multiple minutes within the system. |
| Die/Death | In relation to the entity, it is the removal of its existence within the system and a moving and interacting being. It is the end of said entity. |
| User | The operator of the program at any specific time. |

# Software Requirement

## Use cases

|  |  |
| --- | --- |
| **Name** | **World data gathering** |
| Summary | As the program starts, it will need some information from the user. |
| Rationale | On start of the program, the necessary portions of the simulation will have to be created; otherwise the program will do nothing. |
| Users | Users who begin a simulation |
| Preconditions | The program is launched |
| Basic Course of Events | 1. The user starts the program 2. The software responds by creating a window and asking the user for the type of environment and number of AIs 3. The user uses the new window to enter the above values 4. When the user is satisfied, user enters values 5. The software then uses the values to create a world and *n* number of AIs 6. The simulation begins. |
| Alternative Paths | The procedure can be canceled by closing the software at this point. |
| Post conditions | Internal parameters are saved for future world generation. |

|  |  |
| --- | --- |
| **Name** | **World initialization** |
| Summary | Gathers all elements necessary for the simulation, builds the simulation, and starts it |
| Rationale | After the initial data gathering, the simulation will need to be built and started. |
| Users | Users of the program. |
| Preconditions | Type of world is known (data gathering window has been run). |
| Basic course of events | 1. A world is generated based on values from the data gathering window  2. The first chunk is generated based on the world values  3. A number of animals are generated and placed into the first chunk  4. A window is created  5. The first chunk (and everything in it) is drawn to the window |
| Alternative paths | The program crashes |
| Post conditions | Simulation begins |

## Functional requirements

|  |  |
| --- | --- |
| **Name** | **FR1: Initial data gathering** |
| Summary | The world initialization use case requires the user to enter a number of pieces of information to allow the program to create a customized world. |
| Rationale | A program cannot run if it does not know its required initial arguments. Therefore, it is necessary to ask the user for several pieces of information before beginning the simulation, and thereby create a more accurate and useful simulation. |
| Requirements | When a user launches the program, a window will be created, with a text box, dropdown list, and button. The text box will allow the user to input an integer for the number of agents the user wishes to create. The list will allow the user to select the desired environment from a number of pre-selected environments. The button will allow the user to continue once the values are selected.  Here are computational steps required to implement this requirement.   * When the program is started, the program creates a window containing the above elements and searches for a text file containing the values used previously, if it exists. * If the file was found, the fields are pre-filled using the values from the file. If not, they are left blank. * The program waits for the user to continue. * Upon the button press, the program checks the value in the text field to make sure it is a number * If not, the program will open a message box asking the user to correct the value. * When the value is OK, the program will collect the data and close the window |
| References | none |

## Nonfunctional requirements

|  |  |
| --- | --- |
| **Name** | **Animal Generation** |
| Summary | After the number of Animals to be generated is selected, Animals will be generated. |
| Rationale | If there are no AIs, there is nothing to monitor. |
| Requirements | * The world is generated. * A number of animals will be generated on spaces close to the initial chunk’s center tile. * Each one will be placed on its own individual space. * The animals will begin to move about. |
| References | none |

|  |  |
| --- | --- |
| **Name** | **Animal Logic** |
| Summary | After the animals are generated, they begin to act. |
| Rationale | If the animals do not act, the program cannot monitor their behavior. |
| Requirements | * Animals are generated. * The Animals needs will increment based on how long they can last since their last meal or drink in which they had. * The animal will decide on its priority need. * The animal will begin to search within its line of sight for the closest tile that will provide it with sustenance. * The animal will then generate a logical path in which it takes to the item in which it needs. * It begins its movement to the object, one tile per tick.   If the animal reaches its goal or is within range of the ability to perform the task in which it needs to gain sustenance or rest, it will act in that fashion instead of moving.  If an animal does not meet its sustenance needs, it will die. |
| References | none |

# Software Design

The section describes how ArtIn: Survival needs to be integrated with other devices (System level) and how the software modules are integrated (Architecture level), and how each module is designed (Module level).

## System level

ArtIn: Survival requires that the user has a valid updated version of Java 7e installed on their computer. ArtIn: Survival also requires an adequate amount of RAM to allow for the processing of data. Lastly, ArtIn: Survival requires that the user has ArtIn: Survival unzipped and runs the .jar or installed program.

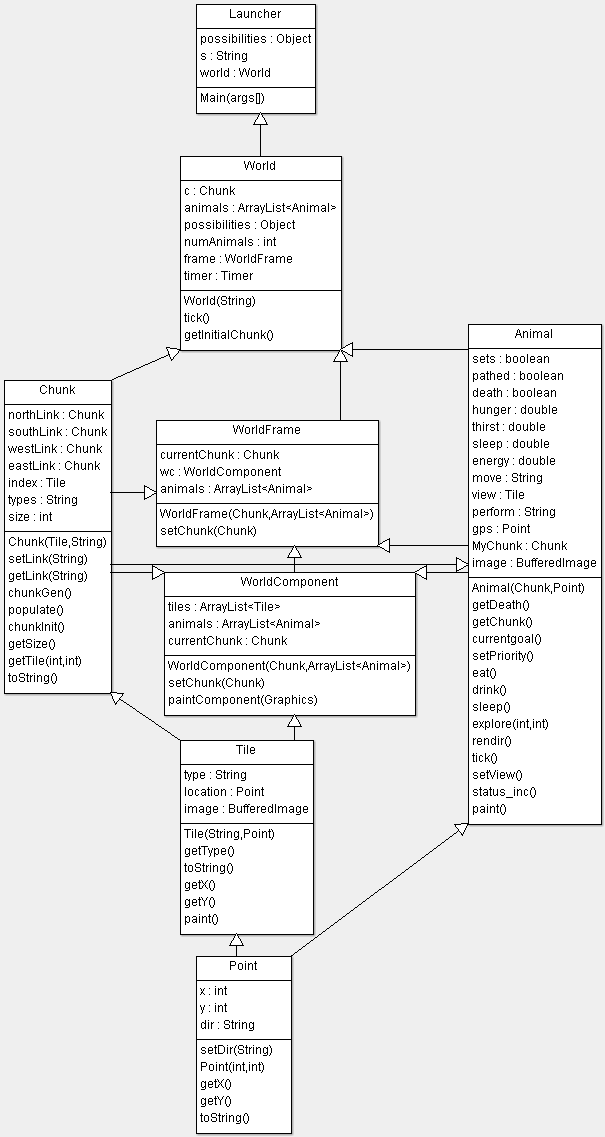
ArtIn: Survival has been tested on Windows (7 and 8) and Linux (tested on RedHat V9 and Ubuntu V13) operating systems, but not Mac OS X. It requires Java 7e.

## Architecture level

ArtIn: Survival has been tested only on x86-64 architecture. ArtIn: Survival may or may not be compatible with ARM architecture.

## Module level

This section details each software component (classes, interfaces, and enums) that is a part of ArtIn: Survival.



UML diagram for ArtIn: Survival

|  |  |
| --- | --- |
| Name | Animal |
| Interface/Attributes | Death- The entities state in the system. (Boolean)  Health- Physical condition of the entity. (Double)  Hunger- The need for food from 1 being full, and 0 being starving. (double)  Thirst- The need for drink from 1 being full, and 0 being dehydrated. (double)  Sleep- The need for sleep from 1 being rested and past 0 being exhausted. (double)  Energy- The total value for all stats being averaged from 1 being completely energized, and 0 being near death. (double)  View- The range of sight on the animal. (Tile[][])  Gps- The entities’ location within a chunk. (Point)  Animal()- Method used to generate Animal.  getDeath()- Used to return death status.  getChunk()- Used to return chunk the animal is within  currentgoal()- Used to set the animals current need it wishes to fulfill  setPriority()- Sets the need that is most crucial to fulfill  eat()- Fills animals hunger from a food tile  drink()- Fills animals thirst from a drink tile.  Sleep()- animal rests until sleep is full  Explore()- The animal travels around until it needs to fulfill a need.  Render()- the act of positioning the direction an Animal faces.  Tick()- Increments one real world second to Animal and begins its actions.  setView()- Sets the Animals view.  Status\_inc()- increments the animals needs one tick lower.  findPath()- paths the animal to a nearby tile to fulfill a need. |
| Hierarchy | Animal is an individual class that does not extend or is extended by any other class. |
| Association | Animal will request from the world(), the tile() information for its surrounding tiles[][], and will then return to the world() its new location for each increment in tick(). |

|  |  |
| --- | --- |
| Name | Chunk |
| Interface/attributes | North/south/east/westLink – the Chunk directly to the north/south/east/west of the current Chunk  Types – array of valid types (string[])  Size – size of the array used to save chunk data (int)  setLink() – used during chunk creation process to set link to another chunk  getLink(String direction) – returns chunk in given direction  chunkGen() – generates and returns a Chunk  populate() – used during Chunk generation to populate a two-dimensional array with tiles  getTile(int x, int y) – returns tile at given coordinates. Cannot account for out-of-index errors. |
| Hierarchy | Chunk is an individual class that does not extend or is extended by any other class. |
| Association | Chunk takes a String type from World and provides tile information to Animal and World. |

|  |  |
| --- | --- |
| Name | Tile |
| Interface/attributes | Type – String of type of tile  Location – holds tile location within the Chunk. (Point)  getType() – returns type string  getX() – returns x coordinate (int)  getY() – returns y coordinate (int) |
| Hierarchy | Tile does not extend or is extended by any other classes. However, Tiles are created solely by the Chunks. |
| Association | Tile takes type and location from Chunk when a new chunk is generated. Tile provides type when the world is drawn and location to World and Animal on request. |

|  |  |
| --- | --- |
| Name | World |
| Interface/attributes | C – holds initial Chunk  Animals – ArrayList of all animals in the World  Frame – frame to draw the World on  Timer – sets the speed of the simulation  actionPerformed() – called each time the Timer reaches its interval, by default 3 seconds. Calls tick function.  Tick() – calls each Animal’s tick function. After, sets active chunk to the focused animal and repaints the frame using the active chunk.  getInitialChunk() – returns the Chunk used in first world generation. |
| Hierarchy | World does not extend or is extended by any other classes. |
| Association | World is created when Launcher requests using the type provided by Launcher. World provides C on request. |

|  |  |
| --- | --- |
| Name | Launcher |
| Interface/attributes | Holds Main method.  Possibilities – Object array of types of world  S – String holding returned type from dialog box  World – World started using S  Main(String[] args) – used to launch the software. |
| Hierarchy | Launcher is the main method of the software. |
| Association | Launcher runs the World. |

|  |  |
| --- | --- |
| Name | WorldComponent |
| Interface/Attributes | Tiles – ArrayList of Tiles. Created when the Chunk to draw to the Frame is unpacked during the constructor  Animals – ArrayList of Animals currently in the world  currentChunk – keeps track of which Chunk is being used  setChunk() – changes currentChunk  paint() – paints each tile, then each animal to the Component |
| Hierarchy | WorldComponent extends JComponent, is not extended by any other classes. |
| Association | WorldComponent takes a Chunk and an ArrayList of Animals from WorldFrame. |

|  |  |
| --- | --- |
| Name | WorldFrame |
| Interface/Attributes | currentChunk – keeps track of Chunk to draw to frame  wc – WorldComponent associated with the frame  panel – JPanel used to organize objects within the frame  setChunk() – changes currentChunk to allow for Animal tracking across multiple chunks |
| Hierarchy | WorldFrame extends JFrame, is not extended by any other classes |
| Association | WorldFrame takes an ArrayList of Animals and a Chunk from World |

|  |  |
| --- | --- |
| Name | Point |
| Interface/Attributes | X – holds int X value in chunk  Y – holds int Y value in chunk  Dir – String to hold direction. Depreciated.  setDir() – sets dir string  getDir() returns dir string  getX() – returns X  getY() – returns Y  setLocation() – sets X and Y variables |
| Hierarchy | Points are only used by Tile and Animal classes. |
| Association | Points are used by objects to keep track of the object’s position in a grid. |

# Software Design

# Software Testing

## Unit tests

Unit tests have not been implemented at this time.

## Integration tests

Integration tests have not been run at this time.

# User guide

## Platforms

As of this moment, ArtIn: Survival works on Windows 7 and 8. It has not been tested on any other platform with a Java 7e installed.

## Installation

ArtIn: Survival is portable and does not need to be installed. It can be run from the source or the jar file.

## User interfaces

There are two main interfaces: the data gathering window and the world window. The user inputs data into the data gathering window. If the data is not entered, the software assumes default values. The world window has two buttons: next and previous. The buttons change the window’s focus to the next and previous animals.

# Issues

## Known bugs

The user cannot yet select a chunk to show, due to a bug potentially caused by the way the chunks are held in memory.

Animals cannot yet move outside of their chunk, due to unfinished code in tile linking.

## Limitations

Full AI calculation is very intensive, and so the software is not as fast yet as it could be. Optimizations have yet to be made. Our software has some of its many features left to be implemented. Because of time constraints and our current workload, some features have not yet been implemented. As of currently, our software does not have links from one tile to the next, making it very difficult and time consuming for the AI to try and map out its surroundings. Because of this missing feature that would require a long drawn out process to be able to complete, we were not able to complete the AI pathing feature which, would allow the AI to move about intelligently in its environment. The code for said program is in place but not implemented or tested. Lastly, we were not able to implement a saving feature because of time constraints. We wanted to have our AI be able to navigate and move about randomly in his world before we made it possible for the session to be saved. So, as of right now, we successfully have our world generated, multiple AIs can be shown and implemented in the world, and they can successfully move about and the frame display what actions they preform.