**1. Technology Introduction**

**1.1 JAVA**

This project uses JAVA as the main developing language. JAVA is released in 1996 and it designed as a pure object-oriented programming language (Anon, 2005). It not only absorbs the various advantages of the C-based language, but also abandons some difficult conceptions of C-based language, such as the pointer type (Waldo & Jim, 1996). JAVA has five primary characteristics which are object-oriented, platform independent, distributed, secured and simple (Anon, 2005). When editing and running a JAVA program, it is necessary to involve these four aspects. Firstly, using text editing software or integrated development environment to define different classes in the JAVA source file. Secondly, compiling source file to generate a binary intermediate code, stored in the class file. Then, using JAVA Virtual Machine (JVM) to run the class file and run the bytecode which generated by the compiler. Last, calling the methods implemented in the class file to implement the calls which from program's JAVA API.

**1.2 GitHub Revision Control**

This project uses GitHub as revision control tool. GitHub is one of today’s popular code revision control tool that it can help developers to track the iteration changes they have made (John et al., 2016). GitHub uses Git which is originally designed for managing the Linux source code (Daniel, 2016). Through using GitHub, developers can save each successive version of their project, therefore they can not only update their project but also rollback their project to past version. In addition, GitHub also facilitates collaboration that it allows developers to work together to develop a project with multiple forks, and eventually submit their own code to the main code base.

**1.3 Maven**

This project uses Maven as a build tool. Maven is a management tool released by Apache used in the Java-based project. It based on project object model (POM) and it can manage project construction, reporting, and documentation with a small piece of descriptive information. Maven helps programmers to build software by minimizing the number of bugs and it can also increase the software reuse (Paul, 2010). Maven includes a Project Object Model, a Project Lifecycle, a Dependency Management System, and a run to define the logic of the plugin target in the lifecycle phase. When programmers use Maven, programmers firstly describe their project with a well-defined project object model, and then Maven can apply crosscutting logic from a set of shared (or custom) plugins.

**1.4 JUnit**

This project uses Junit to the unit testing. Unit testing tests each individual pieces of code, such as functions and methods to help programmers to ensure the correctness of the whole software (Louridas, 2005). JUnit is an open source Java library which has set of classes to help programmers to test their code (Hammell et al., 2002). In Junit, programmers can write *assertions* to verify that the actual output is equal to the expected output rather than write complex *If* statements (Louridas, 2005).

**2. Extra packages requirement**

**3. Build project**

This project uses Maven to build and manage. This project has divided into five parts (client part, common part, core part, shell part and web part), but the current version of the project just uses three parts that the client part and web part can be expanded in the future.

In common part, there are two main packages (config and exception).

In core part, there are five packages (algorithm, config, entity, evolution and util). In algorithm package, the CoreAlgorithm.java is an interface which defines two algorithms for calculating two creatures’ densities. The DefaultCoreAlgorithm.java is a class which implements the interface. In config package, the CoreConfig.java is an interface which defines functions of getting configuration variables and DefaultCoreConfig.java is a class which implements the interface. By this way, this project can update variables which have fixed value just in the params.properties file. If project requires more variables, programmers can just change params.properties file and config methods rather than change code in other places. In entity package, there are two sub packages (creature and environment). In creature package, this project defines an enum class, Species. Species bases on the enumeration variable name to define the method that which entity should have. In addition, in creature package, this project has Hare class and Puma class and they all extend abstract class named AbstractCreature. AbstractCreature implements the Creature.class which is an interface and this interface defines the common function of Hare and Puma classes. The CreatureFactory.class in creature package defines the method to obtain a creature object, therefore, the process of creating creature object is transparent. In environment package, this project defines an enum class, Terrain. The Terrain just has enumerated variables for terrain type. Additionally, in environment package, there are Grid entity and Landscape entity. Grid entity represents every pixel in the map and the Landscape entity represents the whole map which includes the array of Grid. Each grid’s land neighbors count is calculated by Landscape entity. Both Grid entity and Landscape entity have their abstract factory class (GridFactory.class and LandscapeFactory.class) to ensure transparency of creating objects. In util package, this project has a util class for Grid. This util class defines a function to generate halo for Grid array and also calculate the land neighbor count for each grid. In evolution package, this project has a class named LandscapeEvolutionManager used for recording the density state after each iteration.

In shell part, there are five packages (command, config, exception, proxy, service and util).

**4. Run Program**

**5. Test**

This project has tested each function and method through using JUnit assertion test.

**5.1 Common part**

In common part, test mainly focuses on verifying that the method of obtaining the configuration parameter can get the correct parameter value. On Lab machine. Using ‘’ to test.

**5.2 Core part**

**5.2.1 Entity Testing (Creature Testing)**

a. CreatureFactoryTest mainly focuses on verifying that the Creature object can be created successfully. Using ‘’ to test.

b. SpeciesTest mainly focuses on verifying that Species object can get fixed values of species’ parameters correctly. Using ‘’ to test.

c. CreatureTest mainly focuses on verifying that Creature object’s parameters have the correct initial values. Using ‘’ to test.

**5.2.2 Entity Testing (Environment Testing)**

a. GridTest mainly focuses on verifying that Grid object can be created successfully and can add Creature objects into Grid object correctly. Using ‘’ to test.

b. GridFactoryTest mainly focuses on verifying that Grid object can be created successfully by GridFactory object. This test also verifies the correctness of GridFactory’s three constructor functions which with different parameters. Using ‘’ to test.

c. LandscapeTest mainly focuses on verifying that the Landscape object can be created successfully by Grid object which has added halo. Using ‘’ to test.

d. LandscapeFactory mainly focuses on verifying that the LandscapeFactory object can create Landscape object successfully. Using ‘’ to test.

**5.2.3 GridUtil Testing**

GridUtil mainly focuses on verifying that functions in GridUtil are correct. These functions include the function which is used to add a halo to Grid and function which is used to count each grid’s land neighbors. Using ‘’ to test.

**5.2.4 Algorithm Testing**

AlgorithmTest mainly focuses on verifying that the algorithm for calculating grid density changes is correct after one iteration. Using ‘’ to test.

**5.2.5 Evolution Testing**

LandscapeEvolutionManagerTest mainly focuses on verifying that the value of the density change of each grid is correct after multiple iterations. Using ‘’ to test.

**5.3 Shell part**

**5.3.1 Configuration Testing**

ShellConfigTest mainly focuses on verifying that the configuration of density value’s boundary is correct. Using ‘’ to test.

**5.3.2 Service Testing**

ConvertServiceTest mainly focuses on verifying that the map file can be read correctly and Landscape object can be created successfully. Using ‘’ to test.

**6. Summary**

**References**

Daniel van Strien. (2016). An Introduction to Version Control Using GitHub Desktop. *The Programming Historian (2016),* The Programming Historian (2016), 01 June 2016.

Hammell, Thomas, & Nettleton, Robert. (2002). Test first, code later: A guide to integrating JUnit into the development process. (Extreme Programming).(Tutorial). *Java Developer's Journal,* *7*(2), 48.

Java programming language. (2005). *Database and Network Journal,* *35*(5), 11-15.

 COPY THE CITATION TO CLIPBOARD.

John D Blischak, Emily R Davenport, & Greg Wilson. (2016). A Quick Introduction to Version Control with Git and GitHub. *PLoS Computational Biology,* *12*(1), E1004668.

Louridas, P. (2005). JUnit: Unit testing and coiling in tandem. *Software, IEEE,* *22*(4), 12-15.

Paul Krill. (2010). Sonatype links Apache Maven build manager with the Eclipse IDE; The Sonatype Professional suite is built for Java developers and helps minimize bugs while maximizing software reuse. *InfoWorld.com,* InfoWorld.com, Dec 1, 2010.

Waldo, Jim. (1996). Programming with Java. (Technology Information). *UNIX Review,* *14*(5), 31.