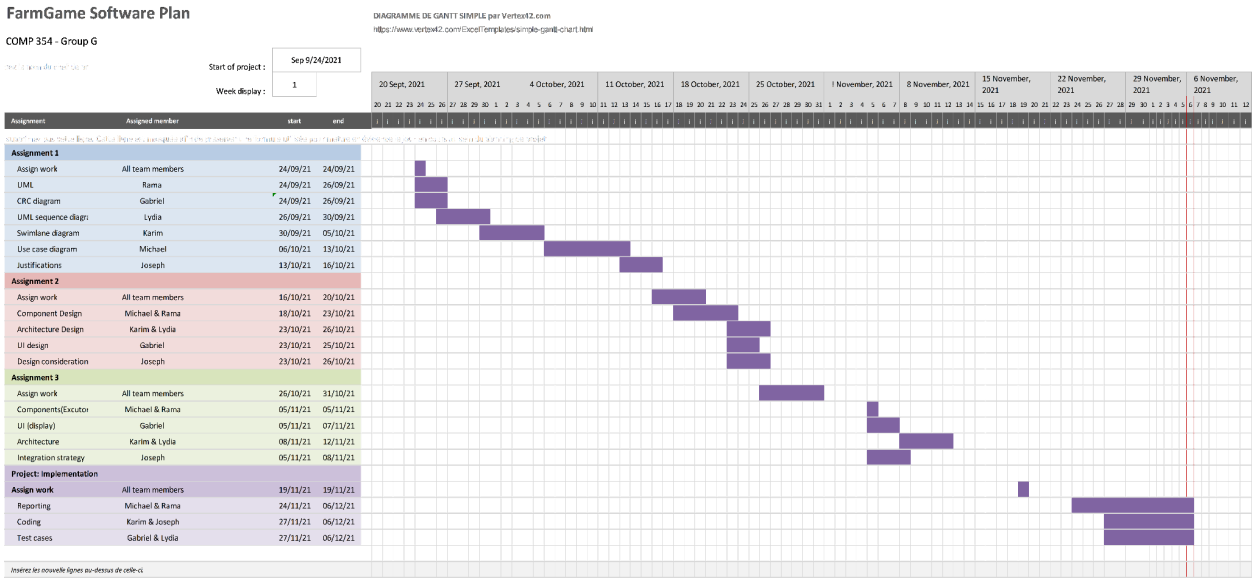
Comp 354 - Section E - Group G
Fall 2021
Final Project

MyBarn/Farm Game Software Planning and Execution

By:
Rama Alrifai (40116096)
Karim Ramy Boshra Botros (40179768)
Joseph Golderger (21958283)
Michael Vincent Orejola (40132872)
Lydia Fodouop (40132543)
Gabriel Martinica (40120255)

I. Prepared Plan



FarmGame Software Plan

COMP 354 - Group G

DIAGRAMME DE GANTT SIMPLE par Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Start of project : Sep 9/24/2021

Week display : 1

Assignment	Assigned member	start	end
Assignment 1			
Assign work	All team members	24/09/21	24/09/21
UML	Rama	24/09/21	26/09/21
CRC diagram	Gabriel	24/09/21	26/09/21
UML sequence diagr	Lydia	26/09/21	30/09/21
Swimlane diagram	Karim	30/09/21	05/10/21
Use case diagram	Michael	06/10/21	13/10/21
Justifications	Joseph	13/10/21	16/10/21
Assignment 2			
Assign work	All team members	16/10/21	20/10/21
Component Design	Michael & Rama	18/10/21	23/10/21
Architecture Design	Karim & Lydia	23/10/21	26/10/21
UI design	Gabriel	23/10/21	25/10/21
Design consideration	Joseph	23/10/21	26/10/21
Assignment 3			
Assign work	All team members	26/10/21	31/10/21
Components(Executor	Michael & Rama	05/11/21	05/11/21
UI (display)	Gabriel	05/11/21	07/11/21
Architecture	Karim & Lydia	08/11/21	12/11/21
Integration strategy	Joseph	05/11/21	08/11/21
Project: Implementation			
Assign work	All team members	19/11/21	19/11/21
Reporting	Michael & Rama	24/11/21	06/12/21
Coding	Karim & Joseph	27/11/21	06/12/21
Test cases	Gabriel & Lydia	27/11/21	06/12/21

Insérez les nouvelle lignes au-dessus de celle-ci.

II. Risk Analysis

An important risk that was encountered before and during the implementation phase of the proposed software was the non-completion of all requirements and tasks presented in previous stages of the project due to time constraints. As there were conflicts in attempting to complete projects and assignments for other courses among group members and to remediate some design and architectural issues, there was about a week left to develop and implement a complete functional software based on our design. This was a challenge for the team, as this required a lot of effort in such a short amount of time. As a result, the proposed software was redesigned and requirements were altered to ensure that a functional software could be delivered by the assigned deadline. Obsolete methods and classes presented in the original design were removed or simplified in the implemented software. For example, the Person class, which initially branched into the now removed Farmer and Visitor subclasses, no longer have the ride(), rest() and restock methods to name a few. Ultimately, the goal was to present a working software that would satisfy old and new requirements.

Another risk found while carrying out the software's implementation was the inconsistent levels of quality for different aspects of the project. Each team member

specialized in different components, which meant it was crucial to separate the project adequately to properly showcase each person's capabilities and produce high-quality software. Three major aspects were defined and were the foundation in dividing the group into subgroups focusing on specific tasks assigned to them. These aspects were coding, testing, and documentation. As they were six team members in total, two people were assigned to each aspect based on their strengths and weaknesses. Members assigned to the coding aspect, for example, were tasked solely on delivering code and implementing the components of the software design. Nevertheless, the coding subgroup would still communicate their thoughts and ideas to the documentation subgroup responsible for completing the project report without having to perform any tasks related to documentation. Thus, by putting into practice the concept of separation of concerns and openly communicating among each other, each implementation task could be executed smoothly and the final product would be up to standard.

III. Implementation

i. Materialized risks and how they were mitigated:

While going through the implementation phase of the software project, it was viable to ensure that the project would be completed by the set deadline. Therefore, in order to complete the software on time, a number of child classes and methods declared in the original design diagrams were removed, many of which are related to the executor component of the software. On average, it would take 30 minutes to write down code for one method, which is equivalent to around 10-20 lines of code. Attempting to implement every subclass and method from the proposed design would prove to be time-consuming and would exceed the time limit that was imposed. To give an example, animal subclasses such as Dog and Horse were removed for simplification purposes. To determine which child classes and methods were to be removed in the final product, detailed analyses were performed on each component of the design. This was to ensure that these components had a reasonable connection with the programming principles that were to be illustrated in the software, more precisely object instantiation, list manipulations, condition statements, if statements, while loops, etc. As a result, it was found that many methods, such as the ride() or guard() methods from the

Animal class, would serve no functional purpose in teaching users about the programming principles mentioned previously. For instance, the move() method in the Animal and Person class would move Person and Animal objects from one location to another. Any methods that would perform the same task were removed as they were deemed too redundant.

It is also worth mentioning how the project implementation was divided among team members, as wrongfully delegating certain tasks to the wrong person would result in subpar software. As mentioned previously, team members were tasked with working on specific aspects of the final project. Because of this, this brought about a risk concerning the coding aspect of the implementation phase, since many major changes were performed. For example, the Feedback mechanism and level system were changed and were added in at the last minute. This great change would normally lead to performing a meeting with all team members to discuss this important deviation from the original design and would greatly deplete time from team members in completing their assigned tasks. As such, any great changes made in the coding aspects were made among team members assigned to the coding subgroup, and these changes were quickly announced to other team members without holding a meeting. Additionally, regarding the testing aspect of the implementation phase, the group agreed on writing and conducting test cases using JUnit, a framework unfamiliar to each group member. Learning about this novel framework would be a time-consuming process if every team member were assigned in creating test cases and it would cause confusion if changes made from one testing component would affect other components as a result. Thus, the testing subgroup, composed of two members, was created to mitigate these issues mentioned and save time to perform other tasks related to the software's implementation.

As a side note, there was also trouble deciding on the name of the software, as the original name of the software was MyBarn. However, the name was soon changed to FarmGame a few days into the implementation of the software. In the end, all group members agreed to stick to this new name despite deviations from the name of the

program from previous assignments, It was worth mentioning this comical short story in case there was any confusion regarding the new name of the software.

ii. **How each component of software design was implemented and tested:**

The test cases are being developed through a testing framework for Java called JUnit. It allows us to test the cases in an external fashion without writing any code in the main program, therefore, the test cases will be run separately from the main software. The cases test for possible errors that might occur during the execution of the software.

The first test case is CowTest.java, which will begin by creating a Cow object using the Cow default constructor. The program will test if the cow object has any milk and if the Cow had milk, then it can be milked using the milk(); however, if the Cow has just been milked, then it can not be milked again and the milk() will fail.

The second test case is InterpreterTest.java, which will test for various possible errors that may occur. This class will test for input sentences that are less than 3 words since the interpreter class requires the input to be at least 3 words. The next part it will test for is including the farmer as the 2nd word in the input sentence; Also, the test class will check for the farmer's name from the input sentence, and if the farmer does not exist the command will fail. It will also test for the animal object in the sentence, which must be the third word in the input sentence; also, the animal must be already created, if the user inputs a nonexistent animal the command will fail again.

Moreover, the user can input to move a farmer or animal to another location, and the test case will check if it is available to move. If the farmer is already in the desired location, and the user tries to move him to the same location, the system will not accept that input and will output a fail message.

Finally, The Level.java class will essentially test if the executor class, it will create a Level, animal, and location objects and will test if the objective of the level object is accomplished.

iii. How components were integrated into a complete software:

The integration of the components took place using Java since, in previous assignments we were asked to follow an object-oriented point of view instead of multiple disconnected subsystems, we constructed a comprehensive functional system that met the criteria. To enable software integration, the code for constructing the program was divided into three independent pieces. The three components are stacked, and the interpreter and display components are wrapper classes for the executor class.

We began by creating the executor part, which includes the structure of the in-game objects. This component in the form of a package is the software's heart since it contains all of the classes and functions that will be called on by the user. This section contains a variety of classes, such as Animal.java, which is subsequently expanded to additional subclasses of animals, allowing the user to play with various methods on various animal objects. We also have the Farmer.java class, which describes one of the most important components; the farmer character is the actor the user will use to accomplish tasks. To wrap all the classes above, we created the Level.java class which has an objective checking method that returns if all animals were taken care of.

Moreover, in order to efficiently show the status of the end-game object, we created the display component. The display class and interface essentially allowed us to print the output of the game’s status and test for errors. This portion is also used to output to the user after execution is over.

Finally, we have the Interpreter part which consists of two correlated classes; the classes will take input from the user in form of simple English sentences since the user will be a child, then the program will split the sentence into keywords that the system knows and will remove all punctuations, Also case sensitivity is deactivated to make the system more receptive and open to see past errors. The interpreter implements built-in classes such as location and animal, which allows the user to have more possibilities to experiment with the software.

IV. GitHub Repository

The developed software can be found on GitHub through the following link. A ReadMe can also be located on the GitHub repository for further details on the implemented software project: https://github.com/riri404/COMP_354_Project.

V. Contributions

Rama Alrifai 40116096 - (Report: Schedule, Implementation/Integration)

Gabriel Martinica 40120255 - (Test Cases and JUnit)

Karim Ramy Boshra Botros 40179768 - (Coding the executor and display)

Joseph Golderger 21958283 - (Coding the interpreter, ASCII art, teaching)

Michael Vincent Orejola 40132872 - (Report: Risk Analysis, Materialized Risks)

Lydia Fodouop 40132543 - (Test Cases and JUnit)