OBJECTP Machine Project Specifications
3rd Term, AY 2016 – 2017

I. Introduction

Create a Java desktop game application derived from the original game called "Plants vs. Zombies". In this game, the objective of the player is to use his limited supply of plants or greens and seeds to prevent a mob of zombies from invading his home.

The environment is a board with 5 or 6 lanes, where each lane has 9 to 10 tiles. Each game lasts for 3 minutes. Once the game starts, sun drops at a constant rate. The player can start collecting sun. Once there is enough sun collected, player can start putting available plants on the tiles. After a few seconds, zombies start to move from one end towards the player's home that is at the other end of the lane. The plants on the lane can either attack or defend against the zombies on that lane. At the last few seconds of the game, a wave of zombies storm to the player's home. At level 1, the number of zombies during the wave is 5. This increases by 2 on each level afterwards.

Game ends when either one of the following conditions is satisfied:

- When one zombie reaches the player's home, Zombies win; or
- When game time ends, Plants win.

II. Zombies

There are a number of types of zombies that differ in attributes: speed, damage, and health. The values for these attributes depends on the type of zombie generated.

Zombies can wear armor (e.g. flag, cone, pole vault, bucket) that may increase their tolerance, or change their movement. Some armors slow down the zombies, and some decreases damage: flag slows down the zombie by 1; the cone decreases speed by 2, decreases damage by 2; and bucket decreases speed by 3, decreases damage by 5.

Some zombies walk, some skip.

	Zombie	Normal zombie Speed = 4, Damage = 10, Health = 70
	Flag Zombie	Moves slightly faster and signals a huge wave incoming
	Conehead Zombie	Uses a traffic cone to protect itself
	Pole Vaulting Zombie	Single jump, jumps over the first plant it encounters with a pole

	Buckethead Zombie	Has a bucket that is extremely resistant to damage

III. Plants

There are a number of types of plants that differ in attributes: sun cost, regenerate rate, damage, health, range, direct damage, and speed.

Sunflower		Gives you additional sun
Peashooter		Shoots peas at attacking zombies
Cherry Bomb		Blows up all zombies in an area
Wall-nut		Blocks all zombies and protects your other plants
Shovel		Lets you dig up a plant to make room for other plant
Potato mine		Explodes on contact but takes time to arm itself
Snow Pea		Shoots frozen peas that damage and slow the enemy

IV. Attributes

Zombie Attributes

Speed	How fast the zombie moves
Damage	How much damage it costs the plant it is attacking
Health	How much damage it can sustain

Plant Attributes

Cost	How much sun needed to use this
Regenerate Rate	How much time it needs to regenerate
Damage	How much damage it costs the zombie
Health	How much damage it can sustain
Range	How far its attack reaches
Direct damage	How much damage it costs when zombie is at a closer range
Speed	How fast the next attack will be

V. Zombie and Plant Generation

Zombies are generated based on the following time intervals:

Period	Interval
30 to 80	Every 10 seconds
81 to 140	Every 5 seconds
141 to 170	Every 3 seconds
171 to 180	Wave of zombies

There is an equal probability for the Zombie to appear in each lane.

Plants are generated depending on their regenerate rates. Plants can be used as long as enough sun is collected by the player.

VI. Minimum Requirements

a. Graphical User Interface

You are to provide a windows application with a mouse interface.

b. Game Phase

Plants and Zombies

During the game, every plant that is available should be clearly indicated.

You are required to implement the following:

Zombie
Flag Zombie
Conehead Zombie
Sunflower
Peashooter
Cherry bomb

For reference and other zombies and plants, you may refer to:
http://plantsvszombies.wikia.com/wiki/Gallery_of_zombies
and http://plantsvszombies.wikia.com/wiki/Plants_(PvZ)

Levels

You are to implement at least three different levels, i.e. three different maps. You may add the number and types of zombies, and the available plants. But all the listed zombies and plants should be available in the final project.

VII. Milestones

a. First Milestone (Phase 1 – due June 28, first 10 minutes of class time)

1. UML Class Diagrams for Zombie, Sunflower, Pea Shooter
2. Implementation of the class Zombie, Sunflower, Pea Shooter
3. Driver to test the basic game (i.e., 1 map). That is, it should cover generation of zombie objects, sunflower objects, peashooter objects at the proper intervals and at the proper tile. No GUI display is expected. However, displays must be made in the Console). For example: “Zombie appeared in Row 1, Column 8 with the following initialized values in the attributes…”. Allow user inputs to be given via keyboard (where pea shooters are

going to be placed, allow collection of sun). For example, “Have enough sun, do you want to add Sunflower or Peashooter to the board?”; “Enter row # then column # where pea shooter will be placed:” Lastly, show per [simulated] time interval, what is happening. For example: “Time: 00:26…. Sun Generated, Zombie previously in Row 1 Column 8 now moved to Row 1 Column 7”.

b. **Final Deliverable (Phase 2 – due August 2, first 10 minutes of class time)**
   1. UML Class Diagrams (Object-Oriented)
   2. GUI with Mouse-Controlled Inputs.
   3. Complete implementation of the game, following Model-View-Control (MVC).

VIII. Submission

1. The following deliverables for **each** phase are to be placed inside a properly labeled short brown envelope:
   a. Signed printout of declaration of original work (Declaration of sources and citations may also be placed here). Sample declaration of original work is as follows:
   “This is to certify that this project is completely out of our combined efforts in studying and applying the concepts learned. We have designed the solution and implemented the corresponding Java source codes ourselves. Any assistance we got were via …[enumerate; eg., web tutorials, asking help regarding concept and not related in any way to the application, etc.] The program was run, tested, and debugged by our own efforts. We further certify that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.”
   b. Printout of the class diagram following UML notations. The class diagram should exhibit the appropriate object-oriented programming principles, including the relationships.
   c. Printout of javadoc-generated documentation for proponent-defined classes with pertinent information
   d. Printout of source code with proper internal documentation
   e. Printout of the test script following the format indicated below.
   f. Flash drive containing the softcopy of the above source code, javadoc-generated documentation, class diagram, and test script
2. The above description of the program is the basic requirement. Any additional feature will be left to the creativity of the student. Bonus points would be awarded depending on the additional implemented features. These additional features could include new specialized zombies or plants, saving and loading current status of player in files, etc. Depending on the scale of the new feature, additional points will be awarded to the team. However, make sure that all the minimum requirements are met first; if this is not the case then no additional points will be credited despite the additional features.
3. This is to emphasize that you DO NOT need to create your own sprites (background pictures, plant/zombie pictures, etc.) for the game. You can just use what's available on the Internet and just include them into your project, just make sure to cite your sources.
4. For the minimum requirements of this MP, all the requirements written in this document should be present and working.
5. Do not forget to include internal documentation (comments) in your code. At the very least, there should be an introductory comment and a comment before every class and every method. This will be used later to generate the required External Documentation for your

Machine Project. You may use an IDE or the command prompt command javadoc to create this documentation, but it must be PROPERLY constructed.

6. Statements and methods not taught in class can be used in the implementation. However, these are left for the student to learn on his or her own.
7. You are required to create and use methods and classes whenever possible. Make sure to use Object-Based and Object Oriented Programing concepts properly. No brute force solution.
8. This project is at most done in groups of 2. A person may work alone. A person cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and his/her groupmate. Copying other people's work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire OBJECTP course and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward. (Also, what is a measly passing grade compared to a life-long burden on your conscience?)
9. Submission of the project is on the indicated submission deadlines. After the first 10 minutes of the class time (and until the end of the session) of the indicated deadline, the project submission is considered late. Late submission will incur 30 points deduction from the resulting project phase score. After the class, no project will be accepted anymore and will therefore result to 0 for that phase.
10. Incomplete submissions will incur deductions.
11. During the demo, all members of the group should be present. The group should use the machine in the lab for the demo and should know how to generate the bytecode file and to run the said file in the command prompt. Apart from question-answer, it is possible that a demo problem be given to the group as part of the demo.
12. A student or a group who is not present during the demo or who cannot answer questions regarding the design and implementation of the submitted project convincingly will incur a grade of 0 for that project phase.
13. During the MP demo, it is expected that the program can successfully be interpreted into bytecode file and will run. If the program does not run, the grade for that phase is 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) will still be checked.
14. All sources should have proper citations. Citations should be written using the APA format. Examples of APA-formatted citations can be seen in the References section of the syllabus.
15. As part of the requirement, the proponent/s should make pertinent back-ups of his/her/their own project. A softcopy of the final unmodified files (for each phase) should be sent to the proponent/s own email address/es. [Note that the envelope, including the flash drive, may not be returned immediately after the demo.]

SUBMISSION CHECKLIST FOR EACH PHASE

a flash drive, properly labeled with name and section, containing at least the following files:

- The Java source files
- The output of Javadoc
- A copy of the UML Class Diagram
- Test script
- any additional libraries you used for your implementation
- Printout of the following:

- Signed declaration of original work (Declaration of sources and citations may also be placed here)
- The Java source files (set the font size to 10 to save space).
- The output of Javadoc for the classes that you created
- UML Class Diagram
- Test Script for each method in each class that you created, in the following table format (with sample entries):

Note: In general, there should be at least 3 categories (as indicated in the description) of test cases per method (except for setters and getters). There is no need to test user-defined methods which are ONLY for screen design (i.e., no computations/processing; just print/println).

Method	#	Test Description	Sample Input Data	Expected Output	Actual Output	Pass/Fail
isPositive	1	Determines that a positive whole number is positive	74	true	true	P
	2	Determines that a positive floating point number is positive	6.112	true	true	P
	3	Determines that a negative whole number is not positive	-871	false	false	P
	4	Determines that a negative floating point number is not positive	-0.0067	false	false	p
	5	Determines that 0 is not positive	0	false	false	P

- Short brown envelope containing all the above requirements, with 2 copies of the proof of submission. One of the proof of submission is attached at the upper right corner of the back of the envelope (not the side with the flap). Each should indicate the following information:

Names	:
Section	:
Submitted To	:
Received by	:
Date / Time Received	:

- Email the Java source files as attachments to YOUR own email address and upload to your class's upload site (e.g., Canvas or Google Classroom, whichever your teacher says). Take note that the actual submitted work would take precedence over the email back-up, and in the case of discrepancies, the actual submission would be considered.