Concordia University
Department of Computer Science
and Software Engineering

Advanced Programming Practices
SOEN 6441 --- Winter 2016

Project Final Build Grading

1. Project Final Build Description

You must deliver an operational version demonstrating the full capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems and completely implementing specific system features. The code build must not be just separated portions of the final project, but a fully operationally integrated software that can be demonstrated by its operational usage.

The presentation should be organized as follows:

1. Brief presentation of the structure of the project as a whole and the tools used in its development.
2. Demonstration of the functional requirements as listed below under “Functional Requirements”.

You are graded according to how effectively you can demonstrate that the features are implemented. If you cannot really demonstrate the integrated features through execution, you will have to prove that the features are implemented by explaining how your code implements the features and what are the expected integration problems, in which case you may lose some marks, even if your explanations are satisfactory.

During your presentation, you have to demonstrate that you are well prepared for the presentation, and that you can easily provide clear explanations as questions are asked about your understanding of the problem being solved, the structure and functioning of your code, as well as your use of tools.

2. Team Identification

Team	Evaluator	Signature	Date	Time

3. Submission to EAS

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category “*project 3*”. The file submitted must be a **.zip** file containing:

- all your code including JUnit test cases
- architectural design document (see “Programming process” below)
- coding standards document (see “Programming process” below)

4. Grading

Presentation		**5**
Effectiveness, structure and demonstrated preparation of the presentation		2
Knowledge of code base/clarity of explanations		3
Functional Requirements		**25**
Map creation and editing		**1**
User-driven interactive creation of a map as a grid of user-defined dimension with grid elements such as scenery, path, entry point and exit point. Saving/loading/editing/verification of an edited map.		1
Game play		**14**
Game starts by user selection of a previously user-saved map, then loads the map.		1
Wave-based play: First (pre-wave phase) the player can place new towers, upgrade towers, sell towers, and signify that critters are allowed in on the map, when all critters in a wave have been killed or reached the end point, a new wave starts.		1
End of game, e.g. when a certain number of critters reach the exit point of the map, or the critters steal all the player's coins, or the player succeeds in killing a certain number of waves.		1
Implementation of currency, cost to buy/sell a tower, and reward for killing critters.		1
Critter waves are created with a level of difficulty increasing at every wave. Difficulty must involve increasing critter speed and toughness.		2
Implementation of at least three different kinds of towers that are characterized by special damage effects. The mandatory special damage effects are: splash (inflicts damage to critters around the targeted critter), burning (inflicts damage over time after a critter has been hit), freezing (slows down the critter for some time).		3
The towers can target the critters using the following mandatory strategies: nearest to the tower, nearest to the end point, weakest critter, strongest critter. It must be possible to set a different targeting strategies for individual towers.		3
Tower inspection window that dynamically shows its current characteristics, allows to sell the tower, increase the level of the tower, select the tower's targeting strategy and view the individual tower's log (see below).		1
Critter observer that allows to dynamically observe the current hit points of any critter on the map.		1
Game management		**10**
Game log that records all events happening in the game, including placement/upgrade/selling of towers, critter wave creation, etc. The log must allow for the viewing of the whole log in sequential order (i.e. ordered in time) or certain portions of the log related to a certain aspect of the game, also ordered according to time, e.g. - Individual tower log: time-ordered log of all events related to a specific tower - Collective tower log: time-ordered log of all events related to all towers (i.e. time inter-meshing of the previous) - Wave log: all activities that happened in any specific wave of the game (select a wave and list sorted by time the events happened in this wave, from pre-wave tower edition phase to end of the wave). - Global game log: all events that happened throughout the entire game up to now, sorted by time.		4
Map log that records in the map file the time of original creation of the map, when it was edited, when it was played and what was the result of the game every time it was played. When a map is being played, the list of five highest scores is presented before the game starts, as well as when the game ends.		2
Save/Load a game in progress: As a game is being played, allow the user to save the game in progress to a file, and allow the user to load the game in exactly the same state as saved.		4
Programming process		**20**
Architectural design—short 3-4 pages document including an architectural design diagram, and a short but complete and clear description of the architectural design. The architectural design shown in the document must be demonstrated to be corresponding to how the actual code in organized.		2
Design patterns—proper use of at least 4 different design patterns, e.g. observer for map rendering or tower inspection, strategy for tower targeting strategies, singleton for game controller, factory for critter wave creation, decorator for levelling up towers.		5
Software versioning repository—well-populated history with several dozens of commits, distributed evenly among team members, a tagged version is created for build 1, build 2 and build 3.		3
Javadoc API documentation—completed for all file, all class and all methods, including all test classes and test methods.		3
Unit testing framework—at least 50 relevant test cases testing the most important aspects of the code including: - Saving/load a map - Saving/loading a game - Targeting strategies (nearest to the tower, nearest to the end point, weakest critter, strongest critter) - Special damage effects (splash, burning, freezing)		4
Coding standards—consistent use of proper code layout, naming conventions and comments. Clear and concise document describing the coding standards used.		3
Total		**50**