Concordia University
Department of Computer Science
and Software Engineering

Advanced Programming Practices
SOEN 6441 --- Winter 2016

Project Build 2 Grading

Instructions for Incremental Code Build Presentation

You must deliver an operational version demonstrating some capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features. The code build must not be just a "portion of the final project", but rather be something useful with a purpose on its own, that can be demonstrated by its operational usage.

The presentation should be organized as follows:

1. Brief presentation of the goal of the build.
2. Brief presentation of the architectural design of your project.
3. Demonstration of the functional requirements as listed on the following grading sheet.
4. Demonstration of the use of tools as listed on the following grading sheet.

You are graded according to how effectively you can demonstrate that the features are implemented. If you cannot really demonstrate the features through execution, you will have to prove that the features are implemented by explaining how your code implements the features, in which case you may lose some marks.

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 the functioning of your code.

Identification

Team	Evaluator	Signature	Date

Grading

Presentation		**5**
Effectiveness, structure and demonstrated preparation of the presentation.		2
Knowledge of code base/clarity of explanations.		3
Functional Requirements		**30**
Map creation and editing		**5**
User-driven interactive creation of a map as a grid of user-defined dimension.		1
User-driven allocation of grid elements such as scenery, path, entry and exit point.		1
Saving a map to a file exactly as edited.		1
Loading a map from an existing file, then editing the map.		1
Verification of map correctness before saving (all types of incorrect maps).		1
Game play		**20**
Game starts by user selection of a previously user-saved map, then loads the map.		1
User-driven placing of towers on the map, following the game's restrictions.		1
Implementation of currency, cost to buy/sell a tower, and reward for killing critters.		1
Implementation of all towers' level-dependent characteristics: cost to increase level, refund rate, range, power, rate of fire, special effects.		2
Allows the user to start a wave, upon which a wave of critters starts moving from the starting point to the ending point along path cells.		2
Implementation of three different kinds of towers that are characterized by having different special damage effects, e.g. splash, burning, freezing.		3
Towers can shoot at the critters, inflicting damage, and eventually killing them.		2
Towers can target the critters using different strategies, e.g. nearest to the tower, nearest to the end point, weakest, strongest, etc.		3
Wave-based play, i.e. when all critters in a wave have been killed, the player can place new towers, upgrade towers, and start a new wave.		2
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.		1
Tower inspection window that shows its current characteristics, allows to sell the tower, increase the level of the tower, and select the tower's targeting strategy.		2
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 design.		3
Design patterns—proper use of at least 3 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.		5
Software versioning repository— well-populated history with several dozens of commits, distributed evenly among team members. Ttagged version is created for build 1 and build 2.		3
API documentation—completed for all files, all classes and all methods.		3
Unit testing framework—at least 30 relevant test cases testing the most important aspects of the code.		3
Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments.		3
Total		**50**

Notes

Additional Narrative Specifications

Map

The software must allow for the creation of custom maps of variable size to be determined prior to the creation of the map. As stated in the project description, it is advised (for simplicity) that you design the map as a grid, where each grid cell is either (1) a scenery cell where towers can be placed, and where the critters are not allowed to move or (2) a path cell where towers cannot reside, and where the critters are allowed to move. One and only one path cell should be assigned as the entry point of the map and one and only one path cell should be assigned as the exit point of the map. There should be a connected graph consisting of path cells that connects the entry point cell to the exit point cell. The map implementation should be designed to allow the creation of a blank map given length and width, and provide member functions to set any cell to anything it might eventually contain as stated above. Your class should have a method to verify the validity of a map, which verifies the validity of the map according to the above specifications.

Tower

Towers should have the following characteristics: buying cost, refund value, range, power, rate of fire, etc. Different types of towers should be created, whose difference is mainly in their behavior, for example in the effect that their bullets have: direct damage, area of effect damage, slowing, additional damage to specific kinds of critters, etc. At least three different types of towers must be created. A tower can also have different levels, organized sequentially and with gradually increased capacities, that allow it to eventually destroy critters more effectively. A tower can be bought for an initial cost, and be further upgraded to subsequent levels for a certain cost for each level. At any time, it can be sold for a certain amount depending on its level. When operating, a tower first detects potential targets within its range, then selects one target and shoots at it, which then inflicts damage and/or applies special effects on the target critter. Different kinds of towers are characterized by special damage effects, e.g. splash damage (damage inflicted to a critter is also applied to nearby critters), burning damage (damage inflicted continues to be applied for a short time), freezing damage (damage inflicted to a critter slows it down for a short time). The special damage effect may also become more effective as the tower's level increases. The towers also have different targeting strategies that can be selected by the user, e.g. targeting the critter nearest to the tower, the critter nearest to the exit point, the strongest critter, or the weakest critter.

Critter

Individual critters should have the following characteristics: reward, hit point, strength, speed, level, etc. The critter wave generator is a component of the game that is called at the beginning of each wave to create a group of critters whose characteristics are adapted to the level of difficulty of the next wave (assuming that every successive wave is of increasing difficulty). The critter wave is a list of critters that will appear sequentially one after the other on the entry point of the map when the game is played. When it is its turn to act, a critter should determine where it will move, assuming that it knows where the exit point is of the map, then move at a certain speed. A critter may be attacked by towers, which will reduce its hit points, eventually dying if it reaches zero hit points. For every critter killed, the player should get a coin reward proportional to the level of the critter. Once a critter reaches the exit point, it steals coins from the player at a rate determined by the critter's strength.