Concordia University
Department of Computer Science
and Software Engineering

Advanced Programming Practices
SOEN 6441 --- Winter 2017

Project Build 3 Grading
Final Project Demonstration

Instructions for Incremental Code Build Presentation

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 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 will get only partial 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, or your required usage of the tools/techniques.

Identification

Team	Evaluator	Signature	Date

Grading (light gray are from previous builds)

Presentation		5
Effectiveness, structure and demonstrated preparation of the presentation		2
Fluid exposition of knowledge of code base/clarity of explanations		3
Functional Requirements		**30**
Character/map/campaign/item creation/editing		**7**
User-driven creation/editing of a validated *fighter* character following the d20 game rules: level, ability scores, ability modifiers, hit points, armor class, attack bonus, damage bonus, multiple attacks, owned items (chosen from the items saved in the items file). Use of predefined fighter types in the creation of a character (see below).		1
User-driven creation/editing of a validated map of user-defined size by placing elements such as entry/exit door, walls, chest (containing items chosen from the items saved in the items file), and characters (friendly or hostile, chosen from the characters saved in the characters file), saved in a file.		1
User-driven creation/editing of a validated campaign that links the entry/exit doors of maps (chosen from the maps saved in the maps file) as a connected directed graph, saved in a file.		1
User-driven creation/editing of validated items (armor, ring, helmet, etc), saved in a file.		1
Weapon enchantments implemented as a decorator pattern (see description below)		3
Play		**23**
Playing a game starts with the user choosing a character to play with (from any of the characters saved in the characters file), then choosing a campaign to play on (chosen from any of the campaigns saved in the campaigns file). The first map in the campaign is displayed, with the character on the entry point of the map, then the player character can move on the map.		1
When a player character enters a new map, the map is adapted to its level. The level of any non-player character (friendly or hostile) on the map is made equal to the level of the player character. Any item on the map (in chests or owned by non-player characters) is adapter to the level of the player character (e.g. level 1-4, items are +1, level 5-8, items are +2, level 9-12, items are +3, level 13-16, items are +4, level 17 and up, items are +5).		1
The game is implemented using a turn-based mechanism. When the player characters enters a map, all characters (player and NPCs) are registered in a list using a d20 initiative roll to determine their order of play during a round. Each round, each character (player or NPC) is given a turn, during which it can (1) move 3 squares (2) attack once (3) interact with other adjacent game elements.		2
The player character can interact with a chest on the map, which should loot the chest, i.e. the items in the chest are put in the character's backpack.		1
The player character can interact with a friendly non-player character, which allows the user to exchange items with the non-player character. For each item you choose from your own inventory, you receive a random item from the non-player character's inventory.		1
The player can interact with a hostile non-player character on the map. For each such interaction, an attack is done on the non-player character using the d20 combat rules, eventually resulting in the death of a character.		3
The player character can interact with a dead NPC on the map which should loot the non-player character, i.e. the items in the inventory (worn and backpack) of the dead character all are put in the player character's backpack.		1
The player character can interact with the exit door on a map. If the objective of the map is fulfilled, the door is used and the character is moved to the next map on the campaign. The player character goes up a level when exiting a map. The campaign ends when the player character exits the last map in the campaign, at which point the main game options are shown again.		1
Character view implemented using the observer pattern: At any point during the game, the user can select any character on the map (either the player character or any non-player character) to trigger a character observer that shows the characteristics of this character, i.e. the character's ability scores.		1
Inventory view implemented using the observer pattern: At any point during the game, the user can trigger a view of the inventory of the currently selected character (see character view above). The Inventory view allows the user to view the character's worn items slots (one of each: armor, ring, helmet, boots, belt, sword, shield) and backpack. If the selected character is the player character, the inventory allows the user to equip or unequip items, i.e. move items between worn item slots and the backpack. This view should be separate from the character view and should be only be shown on demand. The user should be given the option to close the inventory view, at which point is should disappear.		1
Melee and ranged weapon types used in combat. Attack bonus and damage bonus are correctly calculated according to the weapon type. Ranged weapons can attack within range (not beyond). Melee weapon can only attack in adjacent cells. Cells within range are highlighted during a character's turn.		2
Save a game in progress to a file. Load a saved game and resume play in the same state it was before the save.		3
Logging window clearly showing that the d20 rules are followed during game play, including dice rolls, and all applicable values and modifiers used in the calculations.		2
Strategy pattern for character turn (see description below)		3

Programming process		15
Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders. Proper usage of design patterns as described in the functional requirements.		3
Software versioning repository—well-populated history with many dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the project. A tagged version should have been created for build 1, 2 and 3.		3
API documentation—completed for all files, all classes and all methods.		3
Unit testing framework—at least 40 relevant test cases testing the most important aspects of the code. Must include tests for: (1) wearing items should correctly influence the character's abilities; (2) map validation; (3) character cannot wear more than one item of each kind, (4) character movement on a map, (5) map loading, (6) looting a chest, (7) the Builder pattern correctly assigns ability scores, (8) the weapons' special effects in combat using the Decorator pattern, (9) an attack roll is using all the correct attack modifiers, (10) an attack hits if the attack roll is greater than the armor class of the target, (10) the damage inflicted is using the right damage modifiers, (11) weapons are usable within their effective range, (12) character actions are according to their adopted strategy. There should be a 1-1 relationship between a test class and a tested class. All tests should be runnable using a unique test suite.		3
Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.		3
Total		**50**

Notes

Additional Narrative Specifications

Characters

The baseline of this assignment is to create characters belonging to the *fighter* class. The character constructor must automatically generate the following, based on the level (provided at creation time) and class of the character: (1) ability scores (generated randomly using the 4d6 generation method) and ability modifiers, (2) hit points (based on constitution modifier and level), (3) armor class (based on dexterity modifier and worn armor), (4) attack bonus (based level and strength/dexterity modifiers), (5) damage bonus (based on strength modifier, only for melee weapons). The character must be able to wear one item of each of the following kinds: armor, shield, weapon, boots, ring, belt, helmet, each of which should properly enhance the ability score modifiers of the character. See more details below for items.

Maps

Creation of custom maps of variable size to be determined prior to the creation of the map. It is advised (for simplicity) that you design the map as a grid, where each grid cell is either (1) a default floor cell where a character can move or (2) a wall where a character cannot move, or (3) an occupied cell containing a character, opponent, chest, or a door. The editor should be designed to allow the creation of a blank map given height and width chosen by the user, and provide functionalities to set any cell to anything it might eventually contain as stated above.

Campaigns
A campaign is what is played by a character. A campaign is essentially a connected directed graph where the nodes are maps, and edges represent navigation between the exit door of a map and the entry door of another map. The campaign editor should allow the user to select maps saved in the maps file and connect them to form a directed connected graph.

Items
Items can be of type helmet, armor, shield, ring, belt, boots, and weapon. The character may wear at most one of every type of object, and has a backpack that can contain 10 additional items. Items may be enchanted with a +1 to +5 enchantment bonus that upon wearing will eventually influence one of the character's ability scores (Strength, Dexterity, Constitution, Intelligence, Wisdom or Charisma), armor class, attack bonus, or damage bonus. Different types of items should provide only with a certain possibility of enhancement types as per the table below.

Item	May increase either
Helmet	Intelligence, Wisdom, Armor class
Armor	Armor class
Shield	Armor class
Ring	Armor class, Strength, Constitution, Wisdom, Charisma
Belt	Constitution, Strength
Boots	Armor class, Dexterity
Weapon	Attack bonus, Damage bonus

Character Strategy
Use a Strategy pattern to implement how a character can behaves during any one of its turns. This assumes that the game is implemented using a turn-based approach, i.e. the main game loop maintains the list of characters on the map and call their **`turn()`** method one after the other. During a character's turn, a character is allowed do all of the following: (move) move for a maximum of 3 map cells, (attack) attack any other character on the map, (other interaction) such as looting, using a door, or interactions that are not an attack.

Strategy	Behavior
Human Player	This strategy is for a player character controlled by the user. It requires user interaction for determining where the player moves, what NPC it attacks, and what other interactions it will do during a turn.
Computer Player	This strategy is for a player character controlled by the computer. A computer player character's objective is to go to the next map, i.e. fulfilling any objective that you have defined to finish a map.
Aggressive NPC	This strategy is for enemy NPCs. An aggressive NPC will always run toward the player character and attack it. If it comes near a chest or other NPC while doing so, it will loot the chest and attack the NPC.
Friendly NPC	This strategy is for friendly NPCs. A friendly NPC will wander around the map randomly. If it comes near a chest while moving, it will loot it. If a character using the friendly strategy is attacked, it will change its strategy and become aggressive.

Weapon Enchantments

Use a Decorator pattern to add special enchantments to weapons. Upon creation/edition of a weapon, the user can select one or more of the enchantments listed below (selecting many will have their effect stack). The special effect of the weapon takes effect only when the target receives damage from it. The implementation of the Enchantments with a * can be done by adding a character strategy, i.e. the “Frightened” and the “Frozen” strategy.

Enchantment	Special Effect
Freezing*	Target cannot move for a number of turns equal to the enchantment bonus of the weapon.
Burning	Target takes (5x enchantment bonus) damage for the 3 next turns.
Slaying	Target dies instantly.
Frightening*	Target runs away from character for a number of turns equal to the enchantment bonus of the weapon.
Pacifying	Target adopts the “Friendly NPC” character strategy.

Predefined Fighter Types

Use a Builder pattern to implement the generation of ability scores according to predefined character “kinds”: After all 6 scores have been generated using the 4d6 method, the scores are assigned to an ability depending on the type of fighter that this character is: (1) a “bully” uses brute strength to destroy his enemies, (2) a “nimble” favors dexterity and better armor class to evade blows, (3) a “tank” favors survival by more hit points through a high constitution score.

Type of Fighter	Ability scores in decreasing order of importance
Bully	Strength, Constitution, Dexterity, Intelligence, Charisma, Wisdom
Nimble	Dexterity, Constitution, Strength, Intelligence, Charisma, Wisdom
Tank	Constitution, Dexterity, Strength, Intelligence, Charisma, Wisdom