jviner						89	Base Grade w/o Group
Create a simple role playing game						100	Final Grade with Group
DIcE Rubric						**Deduction**	**Reason**
Documentation (20%)	***Design Plan (5%)***	Is the documentation easy to understand?	1%				
		Is the documentation written in a logical sequence?	1%				
		Does the documentation have clear and concise sections?	1%				
		Does the documentation provide a clear general overview of the project?	1%				
		Does the documentation match any provided templates?	0%				
	Testing Procedures (4%)	Do the testing procedures demonstrate code coverage and ensure that good practices are used in regard to testing?	2%				
		Do all test cases indicate the result of the test (pass/fail)?	1%				
		Does the test report include screen shots where appropriate to indicate test results?	1%				
	Project Writeup (4%)	Does the writeup document challenges and surprises encountered during the project?	1%				
		Does the writeup document any lessons learned?	2%				
		Does the writeup show growth in understanding of the material?	1%				
	Grammar/Spelling (3%)	Was the documentation free of grammatical and spelling errors?	2%				
		Is the document formatting consistent?	0.50%				
		Were citations done appropriately?	0.50%				
	Code Formatting (5%)	Does the formatting of the code adhere to the common style as run through pep8?	1%				
		Are variables/functions named appropriately and enable code readability?	1%				
		Are comments placed appropriately, adhere to the style guide, and enable code readability?	1%			1	hero.py lacking any meaningful doc strings
		Are classes/modules/files named appropriately and enable code readability?	1%				
		Was borrowed code cited appropriately as per the style guide and the instructor?	1%				
Implementation (30%)	***Version Control (7%)***	Is a branch created to address each requirement or feature?	2%				
		Are only functioning branches merged to master?	2%				
		Is the history free of generated/artifact files?	1%				
		Are commit messages informative?	1%				
		Is the master branch free of direct work?	1%			1	2 commits directly on main branch
	Architecture (15%)	Are effective and efficient data structures used for the problem at hand?	4%			2	Hero class should take in parameters for hp and dice - allowing variations of hereos in differenet games Entity defines hp and dice attributes - but Monster class does not utilize them
		Did the user adhere to the design outline?	1%				
		Was the code designed and constructed in a modular fashion?	4%				Excellent job restricting 98% of print statements to a single module 9 Very SRP) monster class has 4 prints that would be better elsewhere
		Were generally sound decisions made with regard to architecture?	4%			2	Very first line in documentation for optparse:Deprecated since version 3.2: The optparse module is deprecated and will not be developed further; development will continue with the argparse module. A little more use of packages helps groupd things visually and leave them within separate namespaces
		Is the code DRY?	2%			2	len(lines) called on lines 68, 82, and 86 of monster.py- when it never changes between calls - call it once into variable and reuse variable certain literal strings such as lines 104 and 113, 154 and 180, 178 and 204 are repeated - store in variable and reuse variable
	Testing (8%)	Were comprehensive and robust test cases constructed to include but not limited to the test cases provided in this document?	4%				
		Are all tests repeatable?	1%				
		Do test cases match those described in documentation?	3%				
Execution (35%)	***Parsing (5%)***	Does the program pass python3 compileall . with no warnings?	5%				
	Safety (10%)	Does invalid input cause the program to crash?	5%				
		Does invalid input cause the program to act inappropriately?	5%			2	A value of 5 on main menu page causes a UnboundLocalError exception
	Requirements (15%)	Were all requirements met?	8%			1	8 levels of indent on line 329 of gameplay.py when 7 was the max allowed
		Were all inputs parsed correctly and yield the correct output?	7%				
	Performance (5%)	Does the program execute in a timely manner?	5%				
		Documentation	***20***			***19***	
		Implementation	***30***			***23***	

		Execution	***35***			***32***	
		Total Points Available	***85***			***74***	
			Total Deductions			***11***	
	Suggested Features	***Feature***	***Area***				
		Write a man(1) page for your program.	Documentation	2%		2	
		Use unittest to write as many tests as possible that can be run automatedly. Put tests in a separate test subdirectory of the project.	Implementation	4%		0	test_hero_hit would need to prove it is 10 before hit and 9 after to test why you got 9 - otherwise it reads has hit should always return 9 (2) file appears with 3 functions in the test class at once - not proof of "TDD" 3 step process - this is NOT TDD
		Add support for a type of loot called attack potion. When carried, the list of options in the battle menu should now show an additional action of "drink potion". This does not trigger a monster counterattack. The potion is then removed from the hero's loot, and only the next attack that the hero makes has one extra die.	Execution	3%		3	
		Add support for running away during battle. The list of options in the battle menu should now show an additional action of "run away". The odds of successfully running away are 10% per health point of the hero. If successful, the battle ends. If unsuccessful, the monster counterattacks with its usual number of dice, and the hero only rolls one die instead of three.	Execution	4%		3	failure should be counter attack from monster - but hero loses abililtiy to attack that round
		Add support for saving the game when not in battle. The list of actions in a non-battle round should now show an additional action of "save game". When selected, the game's state (Hero loot, health, etc.) should be stored in a file ~/.dd. Upon starting the game, that state should be loaded automatically. The file is removed if the hero dies during the game.	Execution	4%		4	
		Add support for a human-editable file ~/.dd_monsters which describes one monster per line in the file. This is the pool of monsters from which the program draws to pit against the adventurer. This means each line will contain: The monster's name, the monster's starting health, and the number of combat dice that the monster rolls. If no such file exists, the program should default to using some internally-implemented monsters (see Requirements).	Execution	7%		7	
						15	Points out of max 15 available for Features
				Group Points	2	11	
	Functional Requirements	***Requirement***	***Area***				
		The project must run on the class Virtual Machine.	Execution				
		The program must display the following every battle round: Monster's name, hero's health, monster's health, list of possible actions (attack).	Execution				
		The program must display the following every non-battle round: Hero's health, loot on floor (only if available), list of possible actions (go to next room, get loot (only if loot is available), show loot (only if any loot is carried), quit)	Execution				
		When the hero attacks, the program should interally roll three dice for that hero and the appropriate number for the monster (different monsters have different numbers of combat dice). Whoever rolled the highest value deals one damage to the other combatant. If the highest value is tied, go to the next highest value, et cetera. If all dice values are tied, nothing happens that combat round.	Execution				
		The program should support a -d flag, which causes the program to print out the dice rolls for each combat round.	Execution				
		Upon defeat, a monster may leave behind loot. The chance of leaving behind loot should be (monster's combat dice + monster's health points) × 10%. The hero may choose to take such loot with them or not.	Execution				
		When the hero dies (falls to 0 health), the game should exit, and print a list of loot the hero was carrying.	Execution				

	Requirement	Area
Requirements	Design plan, test procedure, and writeup documents must be submitted with the project.	Documentation
	Test Cases used must be submitted with the project.	Implementation
	All source code must be submitted to the class version control system by 1159EDT on the due date specified.	Implementation
	All documentation must be submitted to the class version control system by 1359EDT on the due date specified.	Implementation
	At least five different monsters must be implemented.	Implementation
	At least five different kinds of loot must be implemented.	Implementation

	Constraint	Area
Constraints	Make use of appropriate variable names.	Documentation
	All documentation must be in PDF format.	Documentation
	PEP-8 code style is required.	Documentation
	Docstrings must be used appropriately.	Documentation
	The project should be stored in your assigned VCS account, under the project name dungeon_dudes.	Implementation
	No third-party files/libraries may be used unless signed off by the Program Managers or Instructors.	Implementation
	The hero starts the game with 10 health.	Implementation
	Each logical portion or feature must be built in its own branch.	Implementation
	Merge (do not fast-forward) all commits to branch master and tag releases appropriately.	Implementation
	The default branch to clone should be master.	Implementation
	Code must be DRY when possible.	Implementation
	The project must be written in Python 3.	Implementation
	Program must be invoked as ./dungeon_dudes	Execution
	Program must not crash or get stuck in an infinite loop.	Execution