Milestone One

Code Review

Raymond Agosto

1 Artifact, 3 Categories

1/26/2025

Category 1: Software Design and Engineering

*Code Existing function*

*Code Analysis*

*Planned Enhancements*

Category 2: Algorithms and Data Structures

*Code Existing function*

*Code Analysis*

*Planned Enhancements*

Category 3: Databases

*Code Existing function*

*Code Analysis*

*Planned Enhancements*

Transcript:

Hello, this is Raymond Agosto, as a student of Southern New Hampshire University with my initial Capstone Code Review. Normally these would be led a group of peers, but in this case I will be reviewing my own work. I would say this presentation fits the Course Outcome #2 which is “Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.” Our purpose here is to go over what the code does, analyze how it works, and determine how it can be done better with planned enhancements. The analysis section will be a list of criteria that I will compare the code to as sufficient or insufficient, but because I’m using one artifact for all 3 categories of evaluation, I will be going over the analysis as a sort checklist once, and probably only refer back to it where necessary for clarifying the enhancements later on. It just wouldn’t make sense to repeat the same checklist points 3 times for the same body of code.

For some background, this Python program is meant to be a simple text-based game. You navigate a simple square map of 9 spaces, collecting items while avoiding an npc enemy which will follow the player. When you start, you get a little exposition, introduction to the environment, protagonist, and objectives. The rules and game mechanics are laid out before the player has the option to make any choices. The game ends either with victory, having collected all the parts and confronting the enemy, or defeat in meeting the enemy without all of the parts collected, unless the player decides to exit the game early which is an option.

CATEGORY 1

Our first category of evaluation is Software Engineering and Design, so let’s look at the existing functionality first in light of that, and we can demo that as well

...

I’m already seeing a couple of things that I would like to circle back to... This None value and some of the spacing for text.

*[Play game with map pulled up]*

As you can see, the game plays almost like a turn-based Pac-Man, where he’s chased around by ghosts eating the Pellets until he finds the Super Pellet that makes him able to turn around and chase the ghosts instead.

So the None’s and the formatting...

...

Analysis

**Structure:**

Code completely/correctly implement design

Code conform to pertinent coding standards

PEP 8: <https://peps.python.org/pep-0008/>

partially with indents, spacing, comments, not with line length

Ideally thin so can be opened side by side with other things

Code well-structured, consistent style, consistent format

No uncalled for unneeded procedures, unreachable code

No leftover stubs, test routines in code

some tests for background info/debugging

Code cannot be replaced by calls to ext reusable components of lib function

No blocks repeated code that could be condensed into single procedure

Storage use efficient

Symbolics used instead of “magic num” constants or string constants

only magic number is number of turns, easily change MAX\_P\_TURNS

No excessively complex modules that need restructured or split into multi

program all in one file, not very object oriented

Documentation:

Code clearly/adequately documented with maintainable commenting style

comments consistent with each other, using same names for things

All comments consistent with code

Variables

All variables properly defined with meaningful, consistent, clear names

different enough in context: pos\_player, pos\_boss, etc.

could potentially use less abbrev

All assigned variables have proper type consistency or casting

Python does not fuss about types that much

No redundant or unused variables

unimplemented plans, but this version they had been removed

Arithmetic Operations

Code avoids comparing floating-point numbers for equality

no decimals just whole numbers

Code systematically prevent rounding errors

Code avoid additions or subtractions on num with greatly diff magnitudes

numbers less than 10 difference, no risk of flow issues

Divisors tested for zero or noise

currently no division just add/subtract

Loops and Branches

Loops, branches, logic constructs complete, correct, properly nested

gameplay loops nested to ensure no unintended exit or circumvent

Most common cases tested first in IF—ELSEIF chains

not always “most common” but usually most impactful.

checks for end conditions before checking if not endgame condition

All cases covered in an IF—ELSEIF block, including ELSE or DEFAULT clause

Every case statement has default

Loop termination conditions obvious and invariably achievable

loop end conditions are clearly explained and easy to reach,

only way to get stuck is to infinitely enter invalid inputs

Indexes or subscripts properly initialized, just prior to loop

Statements enclosed within loops unable to be placed outside loops

all gameplay contained in loops, basically only init outside

Code in loop avoids manipulating the index var or using it upon loop exit

only one index for # of turns? doesn’t do anything outside loop

Defensive Programming

Indexes, pointers, and subscripts tested against array, record, or file bounds

directions = ('North','East','South','West')

Imported data and input arguments tested for validity and completeness

All output variables assigned

default-deny, makes loop fail until valid cond

Correct data operated on in each statement

Every memory allocation deallocated

Python has its own garbage collection

Timeouts or error traps used for external device accesses

Files checked for existence before attempted access

All files and devices left in the correct state upon program termination

Enhancements

As far as enhancing the program in the category of Software Engineering and Design, I plan to recreate the existing functionality in the Java language. I believe this will be a demonstration of skills in attention to detail, as I will need to replicate the functionality of the existing program within the context of a different language with not only different syntax but also different rules governing definition, garbage collection, memory allocation, etc. I believe this aligns with the 4th course outcome which is “Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.”

CATEGORY 2

Now let’s take a look at the Algorithms and Data Structures category of evaluation. Couple things I want to touch on are player navigation and npc movement. So currently I’m using a Python dictionary as a data structure with nested tuple values that can be compared for selection with our user input. These dictionaries are for the map locations and to a lesser extent the items to be collected. This works in Python, because the dictionary has keys that correspond to the other end of the tuple. The functions related to player movement are formatted so that user input is entered as a search attempt within the dictionary, able to reach inner nested tuples upon entering the right key word.

The movements for the npc enemy are based on comparison and a bit of randomization. First of all, each location in the map grid is assigned a set of coordinates within the dictionary. Through conditional statements, the player coordinates are individually compared to the boss enemy’s. After all that, boss randomly “chooses” to move in one of two directions (horizontal or vertical) that would put him Closer to the player.

For enhancements, I plan to use different data structures for map creation like a linked list with nodes instead of a dictionary, and potentially different algorithm for the adversarial navigation element. More variation in map design would allow for more replay ability, while also requiring the non-playable character to have something like a recursive search to effectively traverse the map without getting stuck. Depth-First Search would give the boss a quick decision time, but depending on the complexity of the map, a Breadth-First Search would give the boss the shortest path to the player. Maybe I’ll start with a Depth-First for an Easy opponent, and Breadth-First for a more Difficult opponent. I could also take the program from being a single class to being more object oriented, with multiple classes with their own attributes and methods that are instantiated for use in the main method.

I believe these enhancements will be a demonstration of creativity, as it is an evolution of the previous design, with more complexity and potential for more sophisticated gameplay strategy...

I believe this aligns with the 3rd course outcome of “Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.”

CATEGORY 3

For the category of Databases... there is not really any database used in the original. Everything is stored as variables during runtime that do not persist after the program closes.

My plan for enhancement is to incorporate a database to store statistics about the game, such as the number of times played, highest score, etc. The game statistics will need to be Created, Read, Updated and Deleted when necessary. As far as individual player profiles, that might be outside the scope of this project as it would require login credentials and a whole sign in mechanism. There is not anything about this game that warrants that level of security, as it’s not a banking app with sensitive financial information, or medical records, personally identifiable information. A more complex game might incorporate save states that can be picked up where the player left off and so forth. Of course, I could do the old arcade style system where users can enter custom names for their high scores that would be displayed from greatest to least after game over.

In that case I would need to user some input validation and error handling to ensure that whatever string the user enters as the high score name is valid and does not contain anything that could attempt SQL injection or any kind of system exploitation. That is probably the extent of the security measures needed for this particular game.

Much like before, I believe this demonstrates attention to detail, in that the planning and organization of a database is a major factor in its later effectiveness in inserting, retrieving, and updating the information it holds. I believe this also aligns with the 4th course outcome of “Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.”

I’ve been talking for a while now, so I appreciate your time and attention throughout the code review. It seems like we’ve hit all the necessary points so I think this is a good place to close. Thank you once again.