I.Part I
Planning

1. Inspection ID: Snack Review

2. Team

Author	Yang
Reviewers	(1) Tuguldur Ts
	(2) Grayson Alroy

3. Documents

Work products	snack.cpp
References	NASA SOFTWARE FORMAL INSPECTIONS GUIDEBOOK
Checklists	

4. Meetings	Date	Location	Start	End
Orientation	29/05/2024	Online	18:30	20:30
Review Meeting	30/05/2024	Online	18:30	21:30

5. Planning Objectives
- References obtained for work product.
- Checklists obtained for work product.
- Moderator is trained in Formal Technical Review procedure.
- Team members agree to proposed times/dates.
- Moderator's quick review yields less than 5 major issues.
- Reviewers understand responsibilities and are committed.

6. Planning Effort: 45 minutes

Orientation

7. Prep. Goals 60 min/pg. * 2 pgs. = 120 prep.min

8. Orientation Objectives
   - ⚪ Reviewers understand scope and purpose of work product.
   - ⚪ Reviewers understand checking process, checklists, and references.
   - ⚪ Work product, references, checklists and checking forms provided.

9. Orient. effort 25 minutes * 2 participants = 50 minutes

Preparation

10. Inspection ID	Snack Review	11. Document	snack.cpp
12. Reviewer ID	109006271	13. Reviewer name	Tuguldur Ts.

14. Critical, Severe and Moderate Issues

Num	*Location*	*Severity*	*Chk/Ref*	*Description*
1	Line 8-9	Minor	Redundant items	Redefining true to 1 and false = 0 is redundant in C++
2	Line 10	Severe	Nonexistent subroutine called	gotoxy is not a standard function in C++ and typically requires platform-specific libraries.
3	Line 15	Minor	Variable-type incorrect	Typo in variable name hOuput should be hOutput.
4	Line 17	Minor	Misinterpretation	The typo in hOuput may cause misunderstanding for someone reading the code
5	Line 19	Moderate	Incorrect item	The visible parameter should ideally be a boolean (true or false), not an integer.
6	Line 23	Moderate	Duplicate logic	hStdOut is assigned twice consecutively with the same value
7	Line 27	Critical	Incorrect item	Function declared to return void but attempts to return CONSOLE_CURSOR_INFO object.
8	Line 4	Moderate	Interface/timing problem	<conio.h> is non-standard and not portable, which can lead to compatibility issues on non-Windows platforms.
9	Line 10-17	Critical	Missing computation	The gotoxy function should return an int but does not have a return statement,
10	Line 5-6	Minor	Redundant items	The inclusion of headers without their functionalities being used leads to unnecessary compilation overhead.
11	Line 32	Critical	Syntax error	Incorrect array definition syntax
12	Lines 35-39	Severe	Data problem	Use of uninitialized variable i in printSnake()
13	Lines 43-51	Moderate	Logic problem	Misplaced coordinates in gotoxy() call
14	Line 54-60	Moderate	Hardcoding	Hardcoded text and positions in printInformation()
15	Line 67	Critical	Logic problem	*Incorrect assignment in condition*
16	Lines 79-80	Critical	Iterating loop incorrectly	*Infinite loop due to decrementing loop variable in*

17	Lines 86-87	Minor	Misinterpretation	*Incorrect use of false for an integer variable*
18	Line 119	Critical	Incorrect item	*Syntax error in printf statement*
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

15. Effort 120 minutes

16. Issues	critical	severe	moderate	minor	author Q's
Totals	6	2	5	5	0

17. Preparation Objectives

- Work product has been completely checked.
- All critical, severe and moderate issues are noted on this form.
- All minor issues and author questions are noted on the work product.

Preparation

10. Inspection ID	Snack Review	11. Document	snack.cpp
12. Reviewer ID	111000176	13. Reviewer name	Grayson Alroy

14. Critical, Severe and Moderate Issues

Num	*Location*	*Severity*	*Chk/Ref*	*Description*
1	Line 3-7	minor	Header Redundancy	Some header declaration might not be needed and may cause confusion and redundancy
2	Line 9-10	minor	Redundancy	in C++ we don't need to declare value 1 for true and value 0 for false, we can use true and false right away
3	Line 22	moderate	Correctness	Initialize ConCurInf to zero using ZeroMemory
4	Line 15	minor	Variable-type	Variable name hOuput should be hOutput.
5	Line 39	severe	Undefined variable	*Variable i is undefined*
6	Line 20	moderate	Readability	Improve parameter readability by using boolean type
7	Line 55	severe	Function Parameter Correctness	Add const qualifier to function parameters for read-only indication
8	Line 69	severe	Error Handling	Implement error handling for speed values, should be if(speed ==10)
9	Line 74	critical	Correctness	Add boundary check for i
10	Line 12-18	severe	Return Type	Missing int return type
11	Line 74	severe	Indexing Error	Update the data type of the index variable for proper array indexing
12	Line 119	critical	Syntax Error	Fix syntax error in printf statement, should be printf("Enter to start.");
13	Line 113	critical	Loop Condition	Update the loop condition to iterate over the snake body correctly, should be for(i=0; i<bodyLenght; i++)
14	Line 99	severe	Condition Check Update	Update condition to check for both lowercase and uppercase 'n', should be if(gameKey == 'n' || gameKey == 'N') break;
15	Line 211	critical	Syntax Error (array access)	Incorrect array access, should be setSite(i, sBody[i-1].x, sBody[i-1].y);
16	Line 144	severe	case break	Prevent fall-through behavior in witch statements by adding break after each case.
17	Line 28	critical	Return type Error	Change return type to void (return value type doesn't

				match the function type)

15. Effort 120 minutes

16. Issues	critical	severe	moderate	minor	author Q's
Totals	6	6	2	3	0

17. Preparation Objectives

- Work product has been completely checked.
- All critical, severe and moderate issues are noted on this form.
- All minor issues and author questions are noted on the work product.

Review Meeting	Aggregate Checking Data R1		R2		R3		Total	unit
18. Prep. Effort	120	+	120	+		=	240	(minutes)
19. #Critical Iss.	6	+	6	+		=	12 (2 duplicate) (10 non-duplicate)	(issues)
20. #Severe Iss.	2	+	6	+		=	8 (1 Duplicate) (7 non-duplicate)	(issues)
21. #Moder. Iss.	5	+	2	+		=	7 (1 Duplicate) (6 non-duplicate)	(issues)
22. #Minor Iss.	5	+	3	+		=	8 (3 Duplicate) (5 non-duplicate)	(issues)
23. #Author Q's	0	+	0	+		=	0	(questions)

24. Consolidated list of critical, severe and moderate issues

Num	*Location*	*Severity*	*Chk/Ref*	*Description*
1	Line 8-9	Minor	Redundant items	Redefining true to 1 and false = 0 is redundant in C++
2	Line 10	Severe	Nonexistent subroutine called	gotoxy is not a standard function in C++ and typically requires platform-specific libraries.
3	Line 15	Minor	Variable-type incorrect	Typo in variable name hOuput should be hOutput.
4	Line 17	Minor	Misinterpretation	The typo in hOuput may cause misunderstanding for someone reading the code
5	Line 19	Moderate	Incorrect item	The visible parameter should ideally be a boolean (true or false), not an integer.
6	Line 23	Moderate	Duplicate logic	hStdOut is assigned twice consecutively with the same value
7	Line 27	Critical	Incorrect item	Function declared to return void but attempts to return CONSOLE_CURSOR_INFO object.
8	Line 4	Moderate	Interface/timing problem	<conio.h> is non-standard and not portable, which can lead to compatibility issues on non-Windows platforms.
9	Line 10-17	Critical	Missing computation	The gotoxy function should return an int but does not have a return statement,
10	Line 5-6	Minor	Redundant items	The inclusion of headers without their functionalities being used leads to unnecessary compilation overhead.
11	Line 32	Critical	Syntax error	Incorrect array definition syntax

12	Lines 35-39	Severe	Data problem	Use of uninitialized variable i in printSnake()
13	Lines 43-51	Moderate	Logic problem	Misplaced coordinates in gotoxy() call
14	Line 54-60	Moderate	Hardcoding	Hardcoded text and positions in printInformation()
15	Line 67	Critical	Logic problem	*Incorrect assignment in condition*
16	Lines 79-80	Critical	Iterating loop incorrectly	*Infinite loop due to decrementing loop variable in*
17	Lines 86-87	Minor	Misinterpretation	*Incorrect use of false for an integer variable*
18	Line 119	Critical	Incorrect item	*Syntax error in printf statement*
19	Line 22	moderate	Correctness	Initialize ConCurInf to zero using ZeroMemory
20	Line 55	severe	Function Parameter Correctness	Add const qualifier to function parameters for read-only indication
21	Line 69	severe	Error Handling	Implement error handling for speed values, should be if(speed ==10)
22	Line 74	critical	Correctness	Add boundary check for i
23	Line 12-18	severe	Return Type	Missing int return type
24	Line 74	severe	Indexing Error	Update the data type of the index variable for proper array indexing
25	Line 113	critical	Loop Condition	Update the loop condition to iterate over the snake body correctly, should be for(i=0; i<bodyLenght; i++)
26	Line 99	severe	Condition Check Update	Update condition to check for both lowercase and uppercase ‘n’, should be if(gameKey == ‘n’ || gameKey == ‘N’) break;
27	Line 211	critical	Syntax Error (array access)	Incorrect array access, should be setSite(i, sBody[i-1].x, sBody[i-1].y);
28	Line 144	severe	case break	Prevent fall-through behavior in witch statements by adding break after each case.

25. Review Meeting Objectives
- ⚪ All reviewers present. List absent reviewer ID’s: _All reviewers present_
- ⚪ All reviewers prepared sufficiently for meeting.
- ⚪ All issues noted by Scriber and understood by Author for rework.
- ⚪ Any problems with inspection process have been noted.

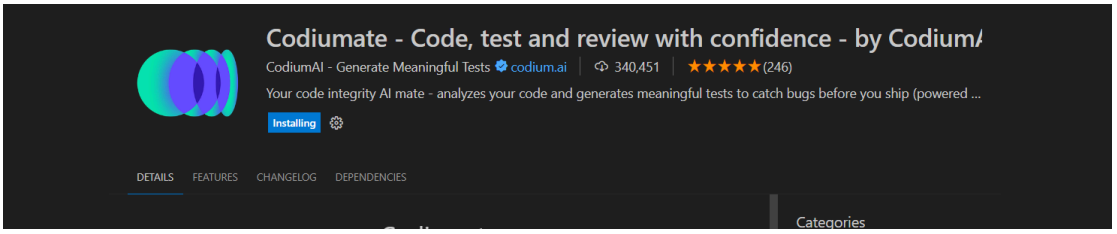
26. R.M. effort 40 minutes * 2 participants = 80 minutes

II. Calculate the Program's Estimated Total Defects, Yield, Defect Density, Inspection Rate, and Defect Finding Efficiency.

- Software Size (in LOC) = **245**
- Total Defects = A + B - C = 18 + 17 - 7 = **28 defects**
  A = Total defects found by the first reviewer = 18
  B = Total defects found by the second reviewer = 17
  C = Total same defects found by both reviewer = 7
- Yield = $\frac{Total\ Defects \times C}{A \times B} \times 100\% = \frac{28 \times 7}{18 \times 17} \times 100\% =$ **64%**
- Defect Density = $\frac{Total\ Defects}{Software\ Size\ (LOC)} = \frac{28}{245} = 0,114\ defects\ /\ LOC$
- Inspection Rate = $\frac{Software\ Size\ (LOC)}{Inspection\ Hours} = \frac{245}{4\ Hours} = 61.25\ LOC/hour$
- Defect Finding Efficiency = $\frac{Defects\ Found}{Inspection\ Hours} = \frac{28}{4\ Hour} = 7\ defects/hour$

III. Open Source Tool to Code Review
Here is our step by step process of using the vscode code review extension. This extension will help us to do a code review and detect whether it has some errors inside the code.

Tool that we use:



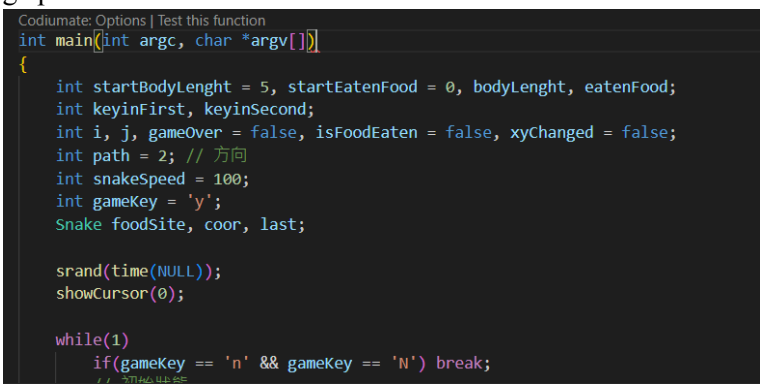
Tool usage and features:

1. **Intelligent Code Completion**: Provides context-aware code suggestions as you type, enhancing productivity by offering relevant options based on your current context and coding patterns.
2. **Automated Code Refactoring**: Offers suggestions and automated tools for improving code readability, maintainability, and performance, helping developers refactor code efficiently while adhering to best practices.
3. **Error Detection and Correction**: Detects potential errors in real-time and provides actionable insights to correct them, reducing debugging time and enhancing code quality.
4. **Code Generation and Snippets**: Generates boilerplate code or code snippets based on your input or specific requirements, speeding up development tasks and reducing repetitive coding.

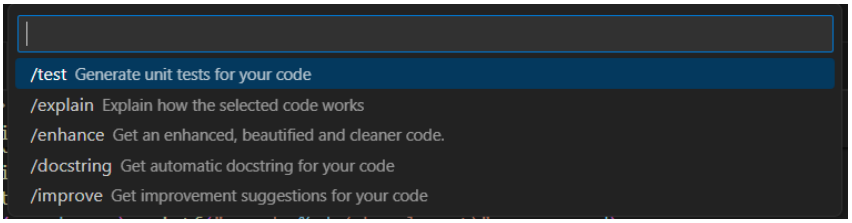
Installation procedure:

1. Open Visual Studio Code
2. Open Extension panel (ctrl+shift+x)
3. Search keywords "code review"
4. Choose the one called "Codiumate by Codium"
5. Install "Codiumate"
6. Sign in and follow the required step to do initialization

Tool usage procedure:

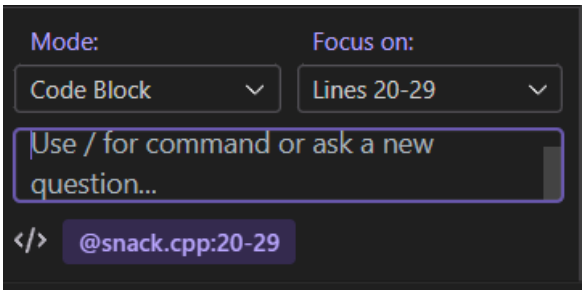
1. 

When we have successfully installed the codiumate extension, there will be Codiumate options on top of each function in our code. (on top of int main)

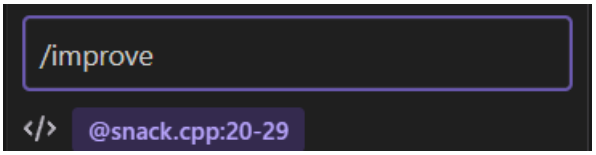
2. 

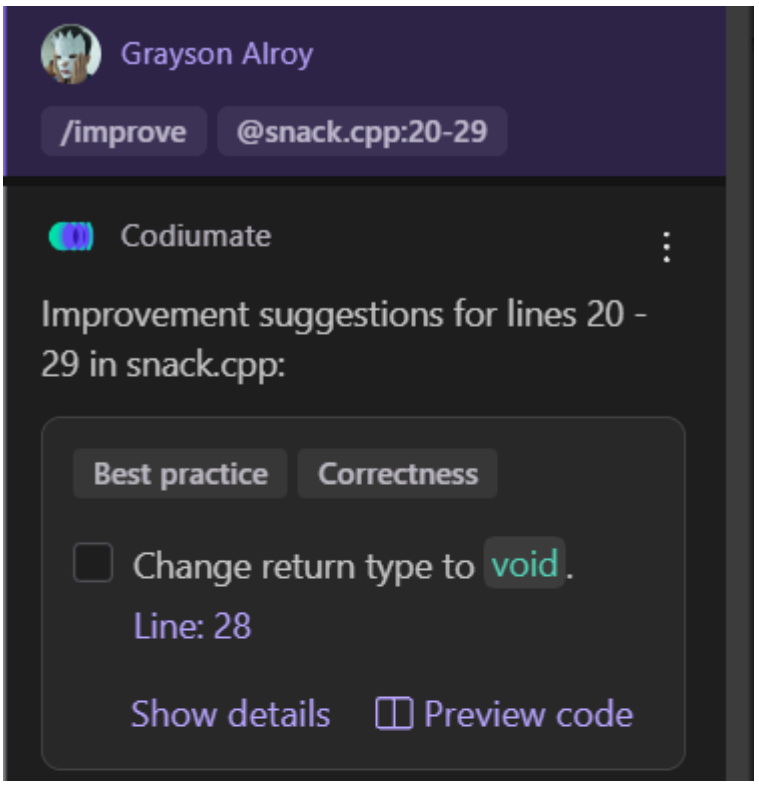
Click on the codiumate options, and it will lead users to these 5 options to help review user's code. It include:
   - /test = Generate unit test for your code
   - /explain = Explain how the selected code works
   - /enhance = Get an enhanced, beautified and cleaner code
   - /docstring = Get automatic docstring for your code
   - /improve = Get improvement suggestions for your code

   Here, since I want to review defects on the code given, I chose **/improve.**

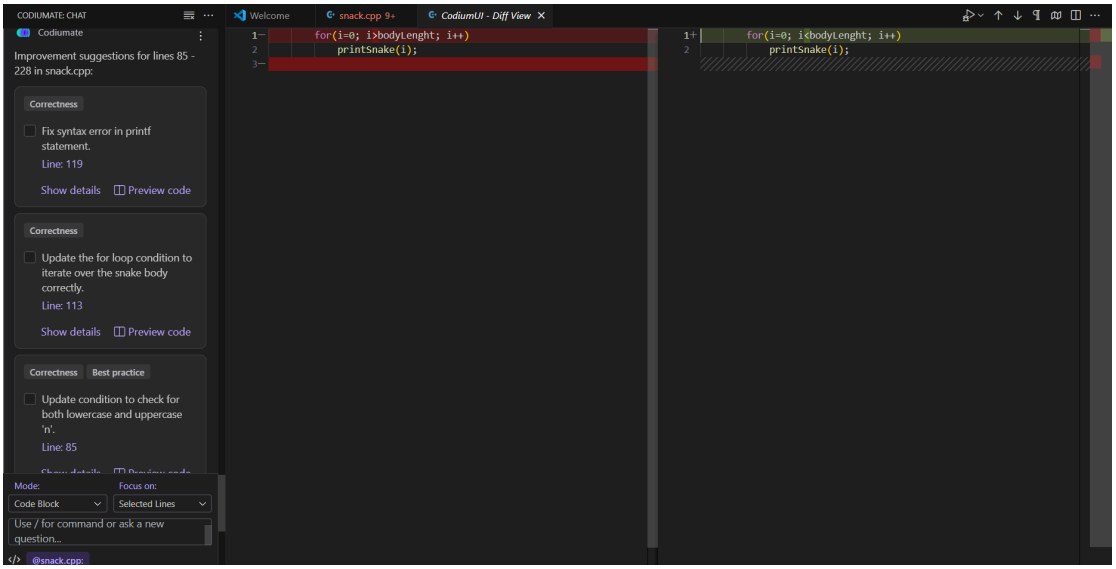
3. 

Type /improve on the command box

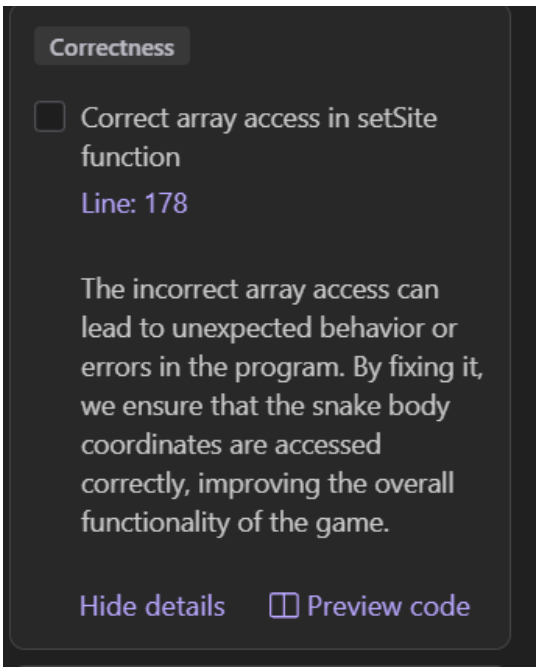
4. 

5. 

Then, codiumate will automatically review our code and give suggestions on which line of our codes need improvement as shown in the screenshot above. It also provides a show details button, and preview code button for the users to look more and dives into detail about the improvement suggestions that is being made.

6. 

When we click at the Preview code button, here is the display: the left side shows our original code, and the right side shows the tool's suggestion on our code.

7. 

   Here is the display when we click at the Show details button, it will explain more about the mistake on our code.

8. We can apply these same methods to all of our code to have an effective, correct, and well-written code.

IV. Please try the ChatGPT-CodeReview tool to assist with code review

ChatGPT-CodeReview Tool: Key Findings

- **Syntax Errors:** Several lines contain syntax errors, such as missing semicolons, incorrect usage of operators, and Python-style syntax in a C++ file.
- **Logic Errors**: Some loops and conditions are incorrectly implemented, which could lead to infinite loops or unintended behavior.
- **Non-Portable Code:** The code uses system-specific functions like system("CLS") and Sleep(), which are not portable across different operating systems.
- **Redundant Code**: Several instances of redundant or unnecessary code, including redundant variable assignments and inefficient clearing of console text.
- **Poor Error Handling**: Lack of error checking and exception handling, particularly after system calls and I/O operations.
- **Magic Numbers:** Extensive use of magic numbers throughout the code that could be replaced with named constants for better readability and maintainability.
- **Resource Management:** Potential issues with resource management, such as not releasing handles or not checking the success of resource acquisition functions.

Manual Code Review: Key Observations:

- **Understanding Context:** What is the purpose of the code? (In this case, simulating a snake game).
- **Readability**: Is the code easy to read and understand?

- **Maintainability:** How easy is it to modify or extend the code?
- **Functionality:** Does the code function as expected?
- **Performance:** Are there any obvious inefficiencies?
- **Security**: Are there any potential security risks?

Since I did not help write the code snack.cpp , I had a hard time trying to understand and find a solution to every problem. But it felt like the problems were mutating because one I try to fix one problem, there are another or two more problems occur.

Comparison and Analysis:

Thoroughness: The ChatGPT-CodeReview tool might be more systematic in identifying specific types of issues like syntax errors and non-portable code. However, it may miss contextual subtleties that a human reviewer would notice, such as the overall design and architecture of the code.

Accuracy: While the tool can accurately pinpoint syntactic and some logical errors, it may interpret some aspects of the code out of context, leading to false positives or negatives. A manual review allows for a deeper understanding and contextual interpretation, which is critical in complex systems.

Efficiency: The tool can scan the entire codebase quickly and identify issues in a matter of seconds, which is significantly faster than manual reviews. However, the time saved in scanning must be balanced with the time needed to verify and interpret automated review outputs.

Discussion:

While automated tools like ChatGPT-CodeReview are invaluable for initial code assessments and can significantly speed up the review process, they do not replace the need for manual review. Automated tools excel at catching straightforward, definable problems but often lack the nuance to evaluate complex logical errors, design patterns, and code maintainability issues. Thus, the best practice would be to use both automated tools and manual review in tandem to ensure code quality, security, and performance.