Final Report

Team Name: KAAR

- Semi-finalist winners I2SSE Contest (C05)

Team Members:

- Karo Bokani - 48502472
- Austin Blencowe - 47666021
- Alan Verghese - 48558893
- Ryan Sambhar - 48411671

Introduction

This project attempts to create an engaging and interactive animation featuring a baker and a tomato. Users can expect a dynamic program where they control the baker's movement within a kitchen environment, with a tomato following our baker, trailing behind and swirling to emulate a rolling movement.

Key Elements:

- **Subject**: A baker, a character that users control using the left and right arrow keys.
- **Object**: The tomato, which spins and trails behind the baker.
- **Activity**: A swirling motion displayed in the tomato.
- **Interaction**: A trailing motion displayed in the tomato.

While the landscape for this animation is endless, our program will focus on a simplistic and controlled kitchen setting, emphasising the importance of system design, planning, and the stages before implementation.
Users will be able to move the baker left and right within the kitchen, experiencing a realistic and dynamic interaction as the tomato spins and trails behind the baker. The program includes collision constraints, ensuring that the baker remains within the screen boundaries and interacts with objects in the background. The trailing tomato will have a delayed response to the baker’s movement, creating a more lifelike and engaging animation.
This project utilises an object-oriented programming approach, separating the code into three main classes (Baker, Tomato, Baguette), allowing for enhanced clarity, simplicity, and reusability. This modular approach is essential for maintaining and reviewing the code, ensuring that each segment can be easily updated or reused. Especially as a group, the possibility of conflicting and unregulated code is heightened, and so maintaining simplicity is integral in mitigating this possibility.

User Manual

Introduction

Welcome to the Baker and Tomato Animation! This guide provides instructions on interacting with the animation and understanding its objectives.

Objective

Whilst interacting with the animation, the baker moves around the space resembling a bakery or kitchen. As the user interacts and moves the baker around the kitchen using the left, right, and up arrow keys, the tomato visually follows the baker, still continuously swirling. When the user presses the spacebar enough times and the baker eats all of the baguette, a new baguette spawns in randomly and appears floating up and down in the kitchen, which the user has to interact with the baker's movement to jump and make collision with the baguette image using the arrow keys. The baker's movement is restricted by the visual aspects of the kitchen environment. For example, the leftmost edge is restricted by the pot, and the rightmost edge is restricted by the bench and countertop. These restrictions add realism to the user's interaction to the animation.

Guide the baker through his kitchen, collect baguettes, and enjoy the company of the tomato.

Controls

- **UP**: Jump
- **LEFT**: Move Left
- **RIGHT**: Move Right
- **SPACE**: Start Eating (when a baguette is collected)

How to Play

Starting the Game:

Note: There is no executable file for the animation therefore, it must be run using the Processing IDE.

1. Ensure that your directory contains the following files:
    - `bakertomato.pde` (Main Animation)
    - `Baker.pde`
    - `Tomato.pde`
    - `Baguette.pde`
    - `kitchen.jpg`
2. Open the `bakertomato.pde` file in the Processing IDE.
3. Start the animation using the compile button which can be found at the top left corner of Processing.

4. Once the program has started, the baker and tomato should appear in the kitchen.

Movement:

1. Use the **LEFT** and **RIGHT** arrows to move the baker.
2. Press **UP** to make the baker jump.

Collecting Baguettes:

1. Guide the baker to touch the baguette to collect it.
2. The baker holds the baguette once collected.

Eating Baguettes:

1. Press **SPACE** when a baguette is collected to eat.
2. Watch the baker enjoy the baguette.

Tomato Companion:

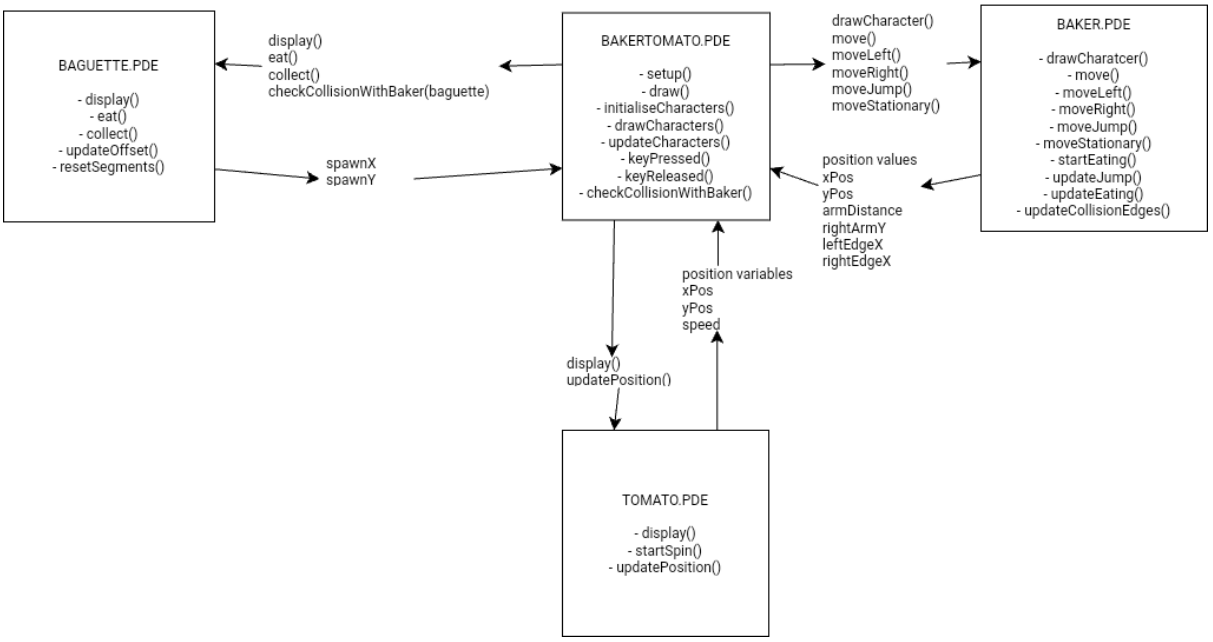
1. The tomato follows the baker.
2. Its position updates based on the baker's movements.

Conclusion

Enjoy the Baker and Tomato Animation! Experience fun adventures in the kitchen, collect baguettes, and savour every moment with the tomato.
If any errors or issues, please refer back to the user manual for support.

Design and Architecture

The class diagram in our Design and Architecture section showcases the main components of our project's architecture, highlighting how classes, `baker.pde`, `baguette.pde`, `tomato.pde`, and `bakertomato.pde` interact. It details each class's operations and attributes to clear up their roles within the system. For example, `Baker.pde` has methods like move() and drawCharacter(), showing interactions and data flow such as position handling and collision detection with other game elements. This visualisation helps grasp the overall structure and flow of the game logic.



User Acceptance Tests

Interim Report User Stories

- List the user stories and requirements that were the source for the user acceptance tests.

Tomato Animation (Interim)

- **As a user**, **I want to** see a red tomato on the centre of the display which swirls clockwise for three seconds when clicked, **so that I** can interact with the animation.

Tomato Animation Acceptance Tests

- **Given that** the program displays a red tomato, **When** the user clicks the tomato, **Then** it swirls clockwise for three seconds.
- **Given that** the tomato has finished swirling, **When** the user clicks it again, **Then** it starts swirling again.
- **Given that** the tomato is displayed, **When** the user does not click it, **Then** the tomato remains static.
- **Given that** the tomato is displayed, **When** the animation starts, **Then** it should be centred on the screen.
- **Given that** the tomato is displayed, **When** the animation starts, **Then** it must be red with a green stem.
- **Given that** the background is displayed, **When** the animation starts, **Then** it should be white.

Baker Animation (Interim)

- **As a user**, **I want to** see a baker in a kitchen that moves left or right when the corresponding keys are pressed, **so that** I can move the baker around the kitchen and I can interact with the animation.

Baker Animation Acceptance Tests

- **Given that** the program displays a baker, **When** the user presses the left arrow key, **Then** the baker moves left.
- **Given that** the program displays a baker, **When** the user presses the right arrow key, **Then** the baker moves right.
- **Given that** the baker is moving left, **When** the left arrow key is pressed, **Then** the baker must not move past the left wall.
- **Given that** the baker is moving right, **When** the right arrow key is pressed, **Then** the baker must not move past the right shelf.

Final Combined User Story

- **As a user**, **I want to** control a baker in a kitchen environment, moving left or right with the arrow keys, making the baker jump with the up arrow key, and eat a baguette with the space bar, **so that I** can interact with the baker in a dynamic kitchen scene with additional animations and interactivity.

Acceptance Tests

- **Given that** the program displays a baker in a kitchen with a trailing, swirling tomato, **When** the user presses the left or right arrow key, **Then** the baker moves in the corresponding direction.
- **Given that** the program displays a baker in a kitchen with a trailing, swirling tomato, **When** the user presses the up arrow key, **Then** the baker jumps.
- **Given that** the program displays a baker in a kitchen with a trailing, swirling tomato, **When** the user presses the spacebar, **Then** the baker starts an eating animation, reducing the size of the baguette and, once finished, spawns a new baguette in a random kitchen location that the baker can collect by jumping.
- **Given that** the program displays a red tomato swirling and trailing behind the baker, **When** the baker moves, **Then** the tomato follows the baker, continuously swirling.
- **Given that** the baker is moving left or right, **When** the baker reaches the left wall or the right shelf, **Then** the baker should stop moving in that direction.
- **Given that** the baker is jumping, **When** the baker reaches the height of a baguette, Then the baker should collect the baguette.
- **Given that** the baker is eating a baguette, **When** any arrow key is pressed, **Then** the size of the baguette should visibly reduce with each press of the space bar.
- **Given that** the baker has finished eating the baguette, **When** the spacebar is pressed, **Then** a new baguette should spawn at a random location in the kitchen.

- **Given that** the baker is moving, **When** the baker moves, **Then** the tomato should always follow the baker’s movements smoothly.
- **Given that** the tomato is swirling, **When** the tomato is in motion, **Then** it should maintain a consistent swirling motion regardless of the baker’s actions.

Acceptance Criteria

1. Movement Control

- The baker must smoothly animate left or right when the corresponding arrow keys are pressed.
- The baker's movement speed should be consistent and realistic, providing a natural feel to the animation.
- The baker must come to a smooth stop when the arrow key is released, without abrupt halts or jerky movements.
- If the baker is in the middle of an action (such as eating or jumping), pressing a movement key should interrupt the current action and initiate movement in the desired direction.

2. Jumping Mechanism

- The baker must execute a believable jump animation when the up arrow key is pressed.
- The jump height should be appropriate, allowing the baker to clear obstacles like baguettes or other characters.
- The jump animation should include anticipation, ascent, and descent phases, providing a sense of weight and gravity to the movement.
- Upon landing, the baker should smoothly transition back into the idle or walking animation, depending on whether movement keys are pressed.

3. Eating Animation

- The baker's eating animation should convey the process of consuming a baguette realistically.
- Each press of the space bar should trigger a smooth progression in the eating animation, with visible changes in the baguette's size.
- As the baguette diminishes in size, the baker's chewing animation should become more pronounced, adding to the realism of the action.
- Once the baguette is fully consumed, the baker should transition seamlessly back into the idle or walking animation.

4. Tomato Animation

- The tomato's swirling animation should be fluid and continuous, conveying a sense of perpetual motion.
- The swirling motion should exhibit variation in speed and intensity, creating visual interest and dynamism.
- As the tomato follows the baker's movements, its swirling pattern should adjust accordingly, maintaining a harmonious relationship with the character's actions.
- Despite changes in the baker's speed or direction, the tomato's swirling animation should remain unaffected, providing a consistent visual element in the scene.

5. Background and Characters

- The kitchen background should feature detailed and immersive elements, evoking the atmosphere of a bustling culinary environment.
- The baker character should be visually distinctive, with unique traits and features that distinguish them from other characters or objects in the scene.
- The tomato should be rendered with vibrant colours and textures, enhancing its visual appeal and ensuring it stands out against the background.
- All characters and objects in the scene should be properly layered and positioned, creating a sense of depth and dimensionality in the animation.

Requirements

User Story: Tomato Animation (Interim)

1. The program must allow users to use their mouse to click the red tomato.
2. The code must create a swirling effect on the red tomato when it is clicked by the user.
3. When clicked, the red tomato should swirl clockwise for three seconds.
4. The user should be allowed to repeat the animation multiple times.
5. The tomato should stay at the centre of the display.
6. The tomato must be red and include a green stem at the top.
7. The background must be white.

User Story: Baker Animation (Interim)

1. The program must display the baker in a kitchen.
2. The character must be easily identified as a baker.
3. The kitchen should be identifiable.
4. The baker must move towards direction depending which key is pressed, right or left key.
5. The baker must not move past the shelf when moving towards the right.
6. The baker must not move past the wall when moving towards the left.

Final Combined User Story

1. The program must initialise with the baker in a kitchen environment.
2. A red, swirling tomato must be present and trailing behind the baker from the start.
3. Implement left and right arrow key functionality to move the baker accordingly.
4. Implement up arrow key functionality to make the baker jump.
5. Ensure the baker cannot move past the boundaries of the kitchen (left wall and right shelf).
6. Implement space bar functionality to trigger the eating animation.
7. Gradually reduce the size of the baguette with each press of the space bar.
8. Spawn a new baguette at a random location once the current one is finished.
9. Implement a swirling motion for the tomato that continues indefinitely.

10. Ensure the tomato follows the baker’s movements, maintaining a trailing position.
11. The tomato must be red with a green stem and must swirl continuously.
12. The baker must be visually distinct as a baker and capable of clear movement animations (walking, jumping, eating).
13. The kitchen background must be easily identifiable as a kitchen with appropriate boundaries.

Acceptance Tests Overview

Test ID	Test Description	User Story	Pass/Fail
AT1	Clicking Tomato Swirls It Clockwise	Tomato Animation	Pass
AT2	Continuous Swirling Motion of Tomato	Tomato Animation	Pass
AT3	Baker Moves Left with Left Arrow Key	Baker Animation	Pass
AT4	Baker Moves Right with Right Arrow Key	Baker Animation	Pass
AT5	Baker Stops at Left Wall	Baker Animation	Pass
AT6	Baker Stops at Right Shelf	Baker Animation	Pass
AT7	Baker Jumps with Up Arrow Key	Final Combined User Story	Pass
AT8	Baker Eats Baguette with Space Bar	Final Combined User Story	Pass
AT9	Tomato Follows Baker's Movements	Final Combined User Story	Pass
AT10	Background Image is Loaded	Baker Animation, Final Combined User Story	Pass
AT11	New Baguette Spawns After Eating	Final Combined User Story	Pass
AT12	Baker Collects Baguette During Jump	Final Combined User Story	Pass
AT13	Tomato Swirls Independently of Baker's Actions	Final Combined User Story	Pass
AT14	Baker Movement Speed Consistency	Final Combined User Story	Pass

AT15	Jumping Animation Anticipation Phase	Final Combined User Story	Pass
AT16	Baker Interacts with Environment Smoothly	Final Combined User Story	Pass
AT17	Baguette Eating Progression	Final Combined User Story	Pass
AT18	Baker Sprite and Animation Consistency	Final Combined User Story	Pass
AT19	Baguette Consumption and Respawn	Final Combined User Story	Pass
AT20	Smooth Transition Between Actions	Final Combined User Story	Pass

System Tests

Test ID	Test Description	Test Steps	Expected Outcome	Actual Outcome	Pass/Fail
ST1	Smooth Baker Movement	Control baker's movement with arrow keys.	Baker moves smoothly without abrupt halts or jerky motions.	Baker movement is smooth.	Pass
ST2	Realistic Jump Animation	Press up arrow key to trigger baker's jump.	Baker executes a believable jump animation with appropriate height and phases.	Baker's jump animation is realistic.	Pass
ST3	Eating Animation Progression	Press space bar multiple times to simulate eating.	Baguette visibly reduces in size with each press, accompanied by baker's arm moving up animation.	Eating animation progresses smoothly.	Pass
ST4	Tomato Swirling Consistency	Observe tomato's motion while moving the baker.	Tomato's swirling animation remains consistent and follows baker's movements smoothly.	Tomato swirling remains consistent.	Pass
ST5	Kitchen Background Detail	Observe the kitchen background for detailed elements.	Background features immersive and identifiable kitchen elements.	Kitchen background is detailed.	Pass

ST6	Baker Movement Interruption	Press arrow keys while baker is eating.	Baker interrupts eating animation and moves in the correspondin g direction.	Baker movement is interrupted correctly.	Pass
ST7	Baguette Spawn Location	Observe new baguette spawn after eating.	New baguette spawns at a random location in the kitchen.	Baguette spawns at random location.	Pass
ST8	Baker Interaction with Baguette during Jump	Jump towards baguette to collect.	Baker collects baguette during jump animation.	Baker successfully collects baguette.	Pass
ST9	Tomato Swirls Independentl y	Observe tomato's motion independent of baker.	Tomato continues swirling regardless of baker's actions.	Tomato swirls independently .	Pass
ST1 0	Baker Movement Speed Consistency	Observe baker's movement at different speeds.	Baker moves consistently regardless of speed.	Baker's movement speed is consistent.	Pass
ST1 1	Jumping Animation Anticipation Phase	Observe baker's jump animation closely.	Baker exhibits anticipation phase before jumping.	Baker's jump animation has anticipation phase.	Pass
ST1 2	Collision Detection with Kitchen Elements	Move baker towards kitchen elements.	Baker collides with kitchen elements and stops movement accordingly.	Baker stops when colliding with elements.	Pass
ST1 3	Baguette Consumption Limit	Continuously press space bar to eat baguette.	Baguette consumption stops after reaching a	Baguette stops shrinking at minimum size.	Pass

			minimum size.		
ST14	Baker Jump Height Limit	Continuously press up arrow key to jump.	Baker reaches maximum jump height and descends realistically.	Baker jumps to maximum height.	Pass
ST15	Tomato Swirling Variation	Observe tomato's motion for speed variation.	Tomato swirling speed varies slightly for visual interest.	Tomato swirling exhibits variation.	Pass
ST16	Baker Movement with Continuous Key Presses	Hold down arrow keys to continuously move baker.	Baker moves continuously in the corresponding direction without interruption.	Baker moves continuously with key presses.	Pass
ST17	Continuous Jumping Animation	Hold down the up arrow key to continuously make baker jump.	Baker continues jumping with a realistic rhythm until the key is released.	Baker jumps continuously with key press.	Pass
ST18	Baguette Consumption Speed	Hold down the space bar to simulate quick baguette consumption.	Baguette shrinks rapidly, reflecting quick consumption.	Baguette shrinks rapidly with each press.	Pass
ST19	Baker Jumping Height Consistency	Press up arrow key repeatedly to observe jump height consistency.	Baker consistently jumps to the same height with each press.	Baker jumps to consistent height.	Pass

ST20	Baker Stops at Kitchen Boundaries	Move baker towards kitchen boundaries.	Baker stops movement when reaching kitchen boundaries.	Baker stops at kitchen boundaries.	Pass
ST21	Baguette Eating and Respawn Limit	Continuously press space bar to eat baguette multiple times.	Baguette consumption stops after reaching a minimum size and new baguette spawns.	Baguette shrinks and respawns correctly.	Pass
ST22	Baker Jumping Limit	Continuously press up arrow key to jump multiple times.	Baker jumps to maximum height each time without deviation.	Baker jumps consistently to maximum height.	Pass
ST23	Tomato Swirling Effectiveness	Observe tomato's swirling effect in different scenarios.	Tomato maintains swirling effect consistently.	Tomato swirling is effective.	Pass

Unit Tests

Baker Movement - Left

Key Elements	Details
Test ID:	UT01
Test Objective:	Verify that the baker moves left when the left arrow key is pressed.
Test Environment:	Processing IDE, `bakertomato.pde` loaded
Test Inputs:	Pressing the left arrow key
Test Steps:	1. Run the program. 2. Press the left arrow key.

Expected Outcome:	Baker moves left on the screen.
Actual Outcome:	Baker moved left as expected.
Pass/Fail Status:	Pass

Baker Movement - Right

Key Elements	Details
Test ID:	UT02
Test Objective:	Verify that the baker moves right when the right arrow key is pressed.
Test Environment:	Processing IDE, `bakertomato.pde` loaded
Test Inputs:	Pressing the right arrow key
Test Steps:	1. Run the program. 2. Press the right arrow key.
Expected Outcome:	Baker moves right on the screen.
Actual Outcome:	Baker moved right as expected.
Pass/Fail Status:	Pass

Tomato Positioning

Key Elements	Details
Test ID:	UT03
Test Objective:	Verify that the tomato moves behind the baker's position.
Test Environment:	Processing IDE, `bakertomato.pde` loaded
Test Inputs:	Baker's position (`xPos: 200, yPos: 215`) and movement (`speed: 4.0`)
Test Steps:	1. Run the program. 2. Observe the tomato's movement.
Expected Outcome:	Tomato follows behind the baker's position.

Actual Outcome:	Tomato positioned correctly behind the baker.
Pass/Fail Status:	Pass

Baguette Collection

Key Elements	Details
Test ID:	UT04
Test Objective:	Verify that the baguette is collected when it collides with the baker's hands.
Test Environment:	Processing IDE, `bakertomato.pde` loaded
Test Inputs:	Baker's hands position (`targetRightArmY, targetRightArmX, rightArmY, rightArmX`), Baguette's spawn position (`spawnX: random(left boundary, right boundary), spawnY: yPos + CONSTANT_SPAWN_Y_OFFSET`)
Test Steps:	1. Run the program. 2. Move the baker to collect the baguette.
Expected Outcome:	Baguette is collected when colliding with the baker.
Actual Outcome:	Baguette collected successfully upon collision.
Pass/Fail Status:	Pass

Swirling Tomato Component

Key Elements	Details
Test ID:	UT05
Test Objective:	Verify that the tomato maintains a continuous swirling motion.
Test Environment:	Processing IDE, `bakertomato.pde` loaded
Test Inputs:	No specific input required
Test Steps:	1. Run the program. 2. Observe the tomato for continuous swirling.

Expected Outcome:	Tomato swirls continuously without interruption.
Actual Outcome:	Tomato maintained a continuous swirling motion.
Pass/Fail Status:	Pass

Software Quality

Zita-PMD Warnings

Project	Warnings	Fixed
`Baker.pde`	TooManyFields `Baker.pde`#L1	This annotation states that the class 'Baker' has too many variables. We addressed this by refactoring the class to integrate several fields into separate objects and determining whether some fields could be turned into local variables. This annotation was implemented to enhance maintainability (modularity, analysability) and performance efficiency (resource utilisation), making the code more manageable and readable.
`Baker.pde`	LongVariable `Baker.pde`#L38	This annotation states that the variable names are too long. We addressed this by refactoring the code to shorten long variable names like '`INITIAL_ARM_BREATHING_SPEED`' to '`ARM_BREATH_SPEED`', thus retaining the meaning while making the code more readable. This was implemented to prioritise maintainability (modifiability, testability), interaction capability (learnability, self-descriptiveness), and functional suitability (functional appropriateness), mitigating any chance of confusion.
`Baker.pde`	ShortVariable `Baker.pde`#L42	This annotation states that the variable names are too short. We addressed this by refactoring the code to replace short variable names like 's' with more descriptive names like 'speed'. This maintains a reasonable variable name length while enhancing interaction capability (self-descriptiveness, learnability) and maintainability (modifiability).

`Tomato.pde`	TooManyFileds `Tomato.pde`#L10	This annotation states that the class 'Tomato' has too many variables. We addressed this by refactoring the code to integrate several fields into separate objects and determining whether some fields could be turned into local variables. This annotation was implemented to improve maintainability (modularity, analysability) and performance efficiency (resource utilisation), making the code more manageable and readable.
`Tomato.pde`	ShortVariable `Tomato.pde`#L10	This annotation states that variables such as 'sz' are too short. We addressed this by refactoring the code to provide succinct yet descriptive variable names like 'size'. This was implemented to prioritise interaction capability (self-descriptiveness, learnability) and maintainability (modifiability, testability).
`Tomato.pde`	DecentralisedEventHandlingRule `System_Test.pde` #L3 and #L4	This annotation is related to event handling and is critical for maintaining proper functionality. We addressed this by modifying the '`simulateMouseClick`' method to take `'mouseX'` and `'mouseY'` as parameters instead of using global event variables. This was implemented to enhance reliability (fault tolerance, faultlessness) and interaction capability (operability), preventing unpredictable behaviour and bugs.

`Baguette.pde`	TooMabtFields `Baguette.pde`#L1	This annotation states that the class 'Baguette' has too many class variables. We addressed this by refactoring the code to integrate several fields into separate objects and determining whether certain fields could be turned into local variables. This annotation was implemented to improve maintainability (modularity, analysability) and performance efficiency (resource utilisation), making the class more manageable and readable.
`Baguette.pde`	ShortVariable `Baguette.pde`#L42	This annotation states that variables such as `'x'` and `'y'` are too short. We addressed this by refactoring the code to replace short variable names with more descriptive names like `'xPos'` and `'yPos'`. This annotation was implemented to improve interaction capability (self-descriptiveness, learnability) and maintainability (modifiability), making the code more manageable and readable.
`Baguette.pde`	LongVariable `Baguette.pde`#38 and L39	This annotation states that the variable names are too long. We addressed this by refactoring the code to shorten long variable names like `'baguetteBreathingSpeed'` to `'breathSpeed'`, retaining their meaning while keeping them at a reasonable length. This was implemented to enhance maintainability (modifiability, testability) and interaction capability (self-descriptiveness, learnability).

From Week 11 onwards the issues found predominantly consisted of excessive class variables, and overly long or short variable names. These errors, identified by the code checker, persisted throughout each separate code, highlighting areas where our code's readability, maintainability, and functionality could be improved. Although, notably, the checker identified event handling problems that could have led to significant issues regarding the reliability of our code that we otherwise wouldn't have been able to rectify. The code checker proved itself invaluable in refining our code's quality and robustness,

highlighting our patterns (excessive class variable, variable name length) that unknowingly could confuse the user.

Project Management

Roles and Responsibilities

To ensure the successful development and delivery of our animation project, each team member had specific roles and responsibilities tailored to their expertise and contributions.

Karo: Product Owner, Project Manager, Animation Integration Specialist, and Documentation Lead
Karo undertook multiple responsibilities, including product ownership, project management, animation integration, and documentation. He ensured effective prioritisation of project requirements, led communication efforts, seamlessly integrated animations, and authored comprehensive project documentation.

- **Product Ownership:**
- Feature Prioritisation: Responsible for prioritising features and requirements based on user needs, project objectives, and stakeholder feedback, guiding development efforts accordingly.
- Requirement Management: Managed project requirements, ensuring alignment with user expectations and business objectives throughout the development process.
- **Project Management:**
- Task Prioritisation: Oversaw task prioritisation and assignment to team members, ensuring alignment with project goals and timelines.
- Meeting Facilitation: Led project meetings, including sprint planning, review, and retrospective sessions, to ensure effective communication and collaboration among team members.
- **Code**:
- Trailing Tomato Functionality: Implemented the logic for the tomato's trailing animation behind the baker, ensuring smooth transitions and realistic swirling motion. This involved updating the `Tomato.pde` and `BakerTomato.pde` files.
- Integration of Animations: Integrated the baker and tomato animations into a cohesive final animation, ensuring synchronised movements and seamless interaction between elements. Files impacted included `Baker.pde, Tomato.pde`, and `BakerTomato.pde`.
- Object-Oriented Programming: Introduced classes into the codebase and utilised object-oriented programming principles to enhance modularity and maintainability. This included creating and updating the `Baker.pde, Tomato.pde`, and `BakerTomato.pde` files.
- Code Review: Conducted thorough code reviews to ensure code quality and adherence to established standards, providing feedback and suggestions for improvement.
- **Documentation:**

- Introduction: Authored the project introduction section, providing context and overview of the animation program, and explaining its objectives and scope.
- Acceptance Tests: Documented detailed acceptance test cases and criteria based on user stories and project requirements, ensuring all functionalities were covered.
- System Tests: Prepared comprehensive system test plans and documented test results to validate program functionality across different scenarios and use cases.
- Unit Tests: Documented unit test cases and ensured test coverage for critical code components to verify their individual functionality.
- Project Management: Oversaw project timelines, task assignments, and progress tracking to ensure timely delivery, coordinating with team members and stakeholders.
- Ethics: Ensured compliance with ethical standards and proper citation of external sources and tools, maintaining transparency and integrity throughout the project.

Austin: Scrum Master, Lead Developer, and Animation Specialist
Austin served as the Scrum Master, Lead Developer, and Animation Specialist. He facilitated scrum meetings, led animation development, including baker movement and background constraints, and significantly contributed to code review and documentation tasks.

- **Scrum Master:**
- Scrum Framework: Chosen as the scrum master for demonstrating strong organisational skills, effective communication, and a good understanding of the scrum framework. Responsible for facilitating the weekly scrum, review, and retrospective meetings.
- Task Management: Ensured team members adhered to previously agreed tasks and procedures, monitoring progress and addressing any impediments to ensure sprint goals were met.
- **Code**:
- Baker Animation: Designed and implemented the baker animation, including walking, jumping, and eating actions, ensuring smooth and realistic movements. This involved updating the `Baker.pde` file.
- Background Constraints Integration: Integrated functionality to constrain the baker's movement within the kitchen background, enhancing realism and immersion. Changes were made to the main codebase.
- Baguette Animation and Eating: Implemented animation and interaction logic for the baguette, including eating action by the baker, updating the `Baguette.pde` file.
- Jumping Movement: Developed the jumping animation for the baker character, ensuring fluid motion and responsiveness. This involved modifications to the `Baker.pde` file.
- Reviewed Unit Tests: Reviewed and provided feedback on unit test cases to ensure comprehensive coverage and effectiveness in verifying individual code components. Additionally, contributed to the writing of unit tests documentation.
- Integration of Classes: Integrated various classes (specifically `BakerTomato.pde`, `Tomato.pde`, `Baguette.pde`) into the codebase to ensure cohesive functionality and interaction between elements.
- Code Review: Provided feedback and conducted code reviews to maintain code quality and consistency across different modules and functionalities.

- **Documentation:**
- User Manual: Created and updated the user manual, detailing program features, controls, and usage instructions to provide a comprehensive guide for users.
- Design and Architecture: Documented the design and architecture of the animation program, outlining component interactions and system flow to facilitate understanding and future development.
- Software Quality Review: Reviewed code and documentation to ensure adherence to software quality standards and best practices, identifying areas for improvement and optimisation.

Alan: Animation Designer and Documentation Specialist
Alan focused on animation design and documentation. He designed and implemented animations, provided valuable feedback on documentation tasks, and contributed to various documentation aspects, including acceptance tests, system tests, and unit tests.

- **Code:**
- Tomato Animation: Designed and implemented the tomato animation, ensuring smooth swirling motion and realistic behaviour, updating the Tomato.pde file.
- Code Review: Reviewed code contributions from team members to ensure consistency, quality, and adherence to coding standards and best practices.
- **Documentation:**
- Acceptance Tests: Reviewed and provided feedback on acceptance test cases and criteria to ensure comprehensive coverage of functionalities and user requirements. Additionally, contributed to the writing of acceptance tests documentation.
- System Tests: Reviewed system test plans and documentation to validate testing approach and completeness, ensuring thorough validation of program functionality. Additionally, contributed to the writing of system tests documentation.
- Unit Tests: Reviewed unit test cases and documentation to ensure adequate testing coverage and effectiveness in verifying individual code components. Additionally, contributed to the writing of unit tests documentation.
- Ethics: Ensured project compliance with ethical standards and guidelines, promoting integrity and responsible conduct throughout the development process.

Ryan: Software Quality Assurance Specialist and Documentation Contributor
Ryan specialised in software quality assurance and documentation. He conducted code refactoring to improve maintainability, assisted with animation implementation, and contributed to documentation tasks, including project introductions and software quality reviews.

- **Code**:
- Code Refactoring: Conducted code refactoring to align with software quality standards and improve maintainability, addressing issues identified during code reviews and testing.
- Assistance with Animation: Provided initial assistance with the implementation of baker and tomato animations, collaborating with team members to resolve technical challenges.
- **Documentation**:

- Introduction: Contributed to the writing of the project introduction, providing context and overview of the animation program.
- Software Quality Documentation: Focused on documenting the software quality assurance processes, including monitoring and addressing Zita-PMD warnings to ensure high code quality and adherence to standards.

Review Processes

Overview

Each group member played a specific role in the review process:

- Karo focused on reviewing and integrating functionality for the trailing tomato animation (`tomato.pde`) and the integration of animations into the combined final animation (`bakertomato.pde`). Additionally, Karo introduced classes into the codebase and utilised object-oriented programming principles for `baker.pde`, `tomato.pde`, and `bakertomato.pde`. Karo also conducted thorough code review sessions for these files, emphasising error handling strategies and parallel programming approaches to enhance robustness and performance.
- Austin led the development efforts, designing and implementing the baker animation (`baker.pde`). He also integrated functionality for baker movement, background constraints, baguette animation, eating animation, and jumping movement. Austin conducted code review for `baker.pde`, ensuring adherence to coding standards and quality guidelines. Furthermore, he reviewed and updated unit tests for code components, focusing on error handling mechanisms and parallel programming techniques to optimise code efficiency.
- Alan was responsible for designing the tomato animation (`tomato.pde`) and ensuring its alignment with project requirements. He conducted code review for `tomato.pde` to identify issues and suggest improvements, including error handling strategies and parallel programming approaches for enhanced reliability and scalability. Alan also reviewed acceptance tests, system tests, and unit tests, providing feedback and recommendations for enhancement.
- Ryan focused on refactoring code to align with software quality standards and best practices. He conducted software quality assurance, including PMD checks and code optimisation, with a specific emphasis on error handling and parallel programming paradigms to improve fault tolerance and concurrency management. Ryan provided initial assistance with baker and tomato animations and contributed to the code review process, emphasising the importance of robust error handling and efficient parallel processing.

KAAR Process

1. Peer Code Review:

- Each member conducted peer review for code changes related to their respective responsibilities.

- Feedback on code quality, readability, and adherence to coding standards was provided.
- Emphasis was placed on error handling strategies and parallel programming approaches to enhance robustness and performance.

2. Unit Testing:
- Group members wrote and executed unit tests for code components assigned to them (`baker.pde, tomato.pde, bakertomato.pde, baguette.pde`).
- Tests were designed to verify the functionality of specific methods and functions within each component, including error handling scenarios.
- Unit tests ensured that individual code components behaved as expected in isolation, even under exceptional circumstances.

3. System Testing:
- Each member participated in comprehensive system testing to evaluate overall system functionality and integration.
- Tests covered various scenarios and user interactions to validate the behaviour of the entire system, including error handling responses and parallel processing capabilities.
- The focus was on identifying issues related to component interactions, data flow, and overall system stability under different conditions.

4. User Acceptance Testing:
- Group members conducted user acceptance tests based on user stories and acceptance criteria defined for the project.
- Test scenarios were designed to validate that the final product met user expectations and fulfilled project requirements, including error handling scenarios.
- Test results were documented, and any discrepancies were addressed to ensure alignment with project goals and user needs.

5. Software Quality Checks:
- Each member conducted software quality checks specific to their assigned code components (`baker.pde, tomato.pde, bakertomato.pde, baguette.pde`).
- PMD checks, static code analysis, and code optimisation were performed to enhance code maintainability and quality, with a focus on error handling and parallel programming paradigms.
- Identified issues were resolved to improve code readability, performance, and adherence to coding standards, especially in error-prone and concurrency-intensive sections.

6. Documentation Review:
- Group members reviewed and updated documentation, including user manuals, design documentation, and software quality documentation.
- Documentation was aligned with the current state of the project, covering all features, functionalities, and implementation details, including error handling strategies and parallel programming considerations.
- Inline comments and method descriptions were added to the codebase to enhance code readability and maintainability, with specific emphasis on error handling mechanisms and parallel processing approaches.

7. Integration Testing:
- Each member ensured smooth integration of code changes into the main project branch without causing conflicts or regressions.
- Integration tests were conducted to verify seamless operation of the integrated system, with all components interacting as expected, including error propagation and parallel execution scenarios.
- Tests covered various integration scenarios to prevent issues and maintain overall system stability under different error conditions.
8. Peer Testing:
- External individuals not part of the project team conducted peer testing to evaluate the functionality and usability of the animation program.
- Test scenarios simulated user interactions to identify potential issues or improvements, providing valuable feedback for enhancement, including error handling suggestions and parallel processing observations.
- Peer testing results were analysed and incorporated into the project to improve its quality and user experience.
9. Scrum Meetings:
- As the Scrum Master, Austin facilitated weekly scrum meetings to review progress, discuss any blockers, and plan tasks for the upcoming week.
- These meetings ensured effective communication and coordination within the team, helping to address any challenges and keep the project on track.

Definition of Done

1. Code Completeness:
- Implemented functionality for desired features, including movement controls (`baker.pde`), eating animation, jumping mechanism, and tomato animation (`tomato.pde`).
- All code files (`baker.pde`, `tomato.pde`, `bakertomato.pde`, `baguette.pde`) must compile without errors.
- Unit tests must be written and passed for all code components.
- Acceptance criteria for each user story must be met, validating that implemented features meet project requirements.
- Refactoring must be completed to optimise code structure and enhance maintainability.
- Documentation, including user manuals, design documentation, and software quality documentation, must be updated to reflect changes.
2. Review Process:
- Peer code review must be performed for all code changes, ensuring code quality and consistency.

- Peer programming enhances collaboration and problem-solving skills by fostering real-time feedback and knowledge exchange between team members, ultimately leading to higher quality code and accelerated learning.
- Feedback from code reviews must be addressed before code is considered complete.

3. Testing:

- Unit tests must be written and passed for all code components (`baker.pde, tomato.pde, bakertomato.pde, baguette.pde`).
- System tests must be conducted to validate overall functionality and integration.
- User acceptance tests must be passed, ensuring that the final product meets user expectations.

4. Software Quality:

- PMD checks must be performed to identify and resolve any code quality issues (`baker.pde, tomato.pde, bakertomato.pde, baguette.pde`).
- Code must adhere to coding standards and best practices.
- Performance optimisations must be implemented to ensure optimal system performance.

5. Documentation:

- Documentation, including user manuals, design documentation, and software quality documentation, must be reviewed and updated.
- Inline comments and method descriptions must be included to enhance code readability and maintainability.

6. Integration:

- Code changes must be successfully integrated into the main project without causing conflicts.
- Integrated system must operate seamlessly, with all components interacting as expected.

Ethics

In order to create our program, our team needed to use external sources to aid us to achieve our goals and meet the expected goals set by the user stories and acceptance criterias.

Source 1:

Background
For the background of our program we use an AI image generator that generates an image that resembles a baker's kitchen which meets the user story and acceptance criteria number 5. The AI service we used to create the background it called 123rf.com.
Link to the website:
https://www.123rf.com/photo_202129181_bakery-shop-counter-with-bread-and-buns-vector-illustration-graphic-design.html?vti=nbmnmj08kwq0q74t1s-1-1&is_plus=1

To get the specific image that helped meet the acceptance criteria and user story was by using specific prompts using Google: Used prompt - "Baker kitchen pixeled" Through this prompt we were able to explore numerous images that would meet our goal, which is that the image should meet our acceptance criteria, user story and look aesthetically pleasing. Austin soon chose the background we currently use since it met our set goals.

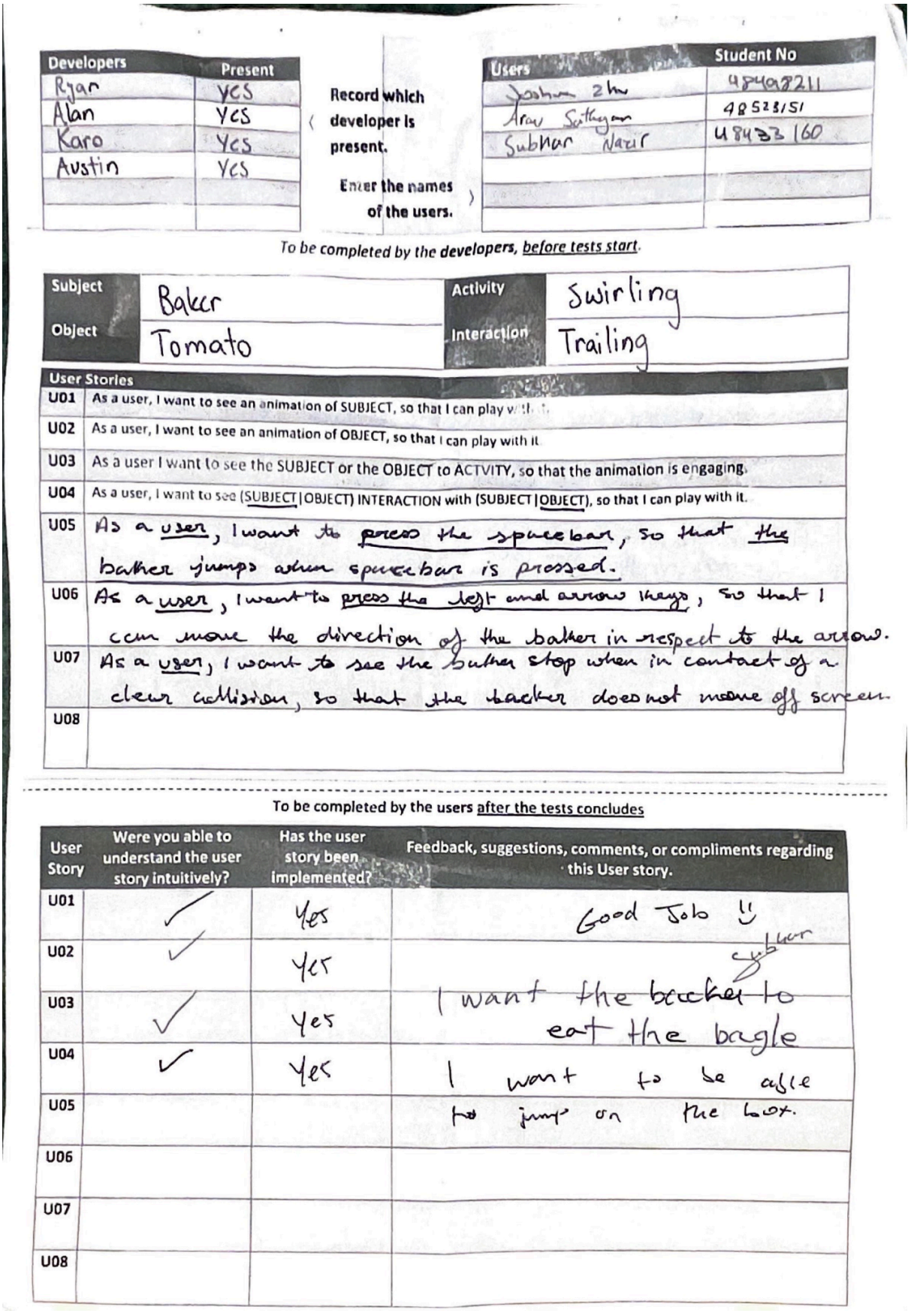
Source 2:

Fixing Code Errors & Creating Classes
As we continuously developed our code and aimed to achieve our goals, we kept running into errors and issues where we needed to resolve how to translate our ideas into code. Our team did overcome the majority of errors we faced, although there were some that were too challenging. That when we used the assistance of AI, more specifically ChatGPT.
Link to website: https://chatgpt.com/
Since we experienced numerous errors, we couldn't keep track of specific prompts that we used to solve our issues. But a general prompt we did use was "why am I seeing and error" through the use of this prompt the AI was able to explain why the error was created and where the error was located. Using this service really helped speed up the process to create our program before the deadline.

Source 3:

Learning Code
For us to achieve the expected outcome of our program as per our set user stories, requirements and acceptance criteria, our team needed to use classes to allow us to have multiple segments of code and create a cleaner architecture which allows our code to be understandable and readable and making it easier to refactor. To achieve this we needed to learn and gain a slightly deeper understanding of what classes are and how to effectively use them in other classes. Our main and only source of gaining this knowledge was from Stack Overflow.
Link to used website:
https://stackoverflow.com/questions/26533350/multiple-classes-in-processing
The specific prompt we used to find the link above was - "stack overflow how to link classes another in processing" This helped us find the blog where an experienced programmer explained how to effectively use classes and how to link them between different files of code.
Outside Contributors
Peer Code Testing
During our allocated workshops, we were given the opportunity to assess programs created by other groups. At the same time, members from different teams were able to assess our code to see if the program achieves our user stories.
The image below is of a worksheet sheet that states our user stories, the user tests and the outcome of our tests when tested by our users (other group members) and any comments by them.

Image 1:



Developers	Present
Ryan	YES
Alan	YES
Karo	YES
Austin	YES

Record which developer is present.

Enter the names of the users.

Users	Student No
Joshua Zhu	48498211
Arav Sathyan	48523151
Subhar Nazir	48433160

To be completed by the **developers**, before tests start.

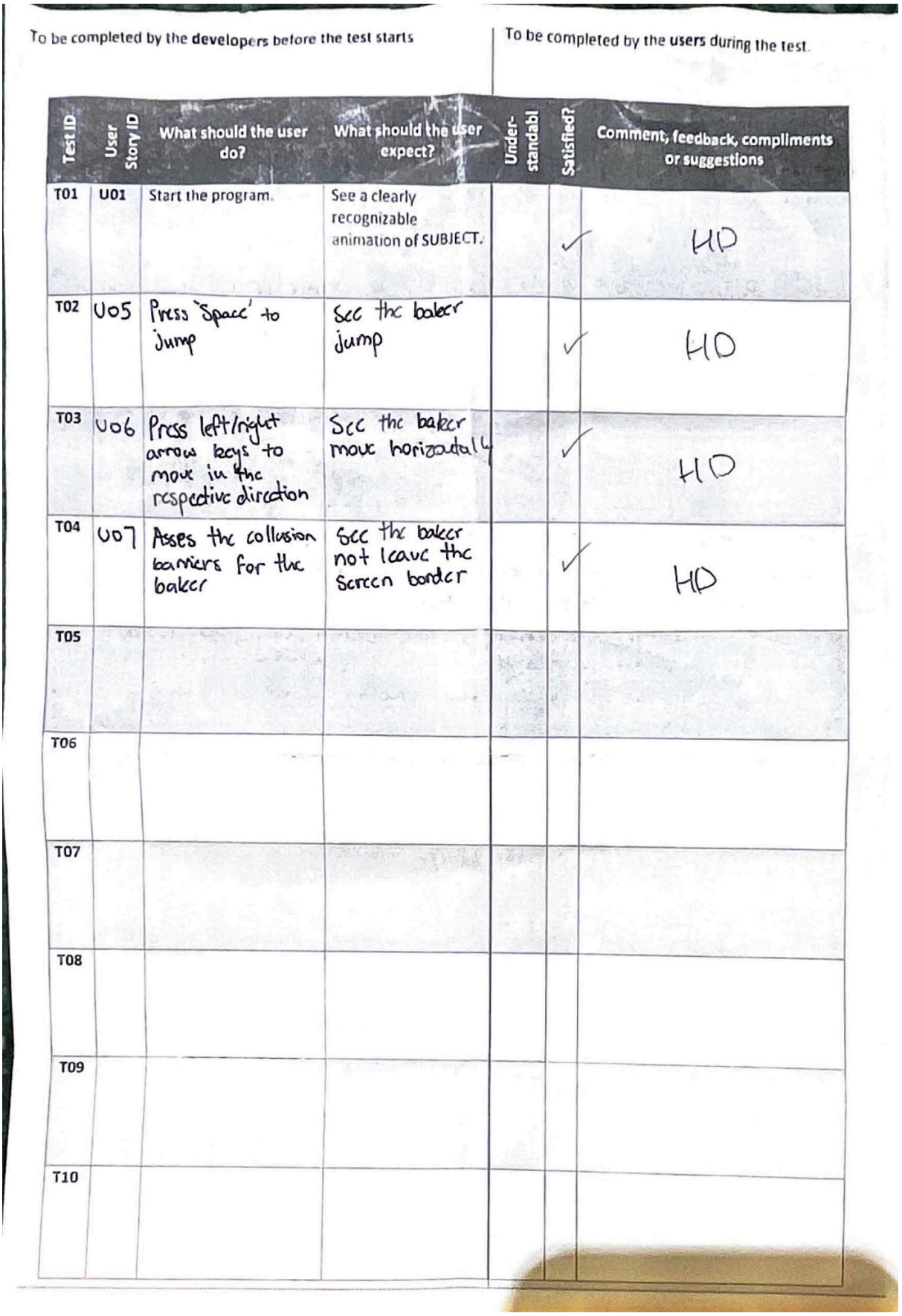
Subject	Baker	Activity	Swirling
Object	Tomato	Interaction	Trailing

User Stories	
U01	As a user, I want to see an animation of SUBJECT, so that I can play with it.
U02	As a user, I want to see an animation of OBJECT, so that I can play with it.
U03	As a user I want to see the SUBJECT or the OBJECT to ACTVITY, so that the animation is engaging.
U04	As a user, I want to see (SUBJECT | OBJECT) INTERACTION with (SUBJECT | OBJECT), so that I can play with it.
U05	As a user, I want to press the spacebar, so that the baker jumps when spacebar is pressed.
U06	As a user, I want to press the left and arrow keys, so that I can move the direction of the baker in respect to the arrow.
U07	As a user, I want to see the baker stop when in contact of a clear collision, so that the backer does not move off screen.
U08	

To be completed by the users after the tests concludes

User Story	Were you able to understand the user story intuitively?	Has the user story been implemented?	Feedback, suggestions, comments, or compliments regarding this User story.
U01	✓	Yes	Good Job !!
U02	✓	Yes	cyber
U03	✓	Yes	I want the backer to eat the bagle
U04	✓	Yes	I want to be able to jump on the box.
U05			
U06			
U07			
U08			

Image 2:



To be completed by the **developers** before the test starts

To be completed by the **users** during the test.

Test ID	User Story ID	What should the user do?	What should the user expect?	Under-standabl	Satisfied?	Comment, feedback, compliments or suggestions
T01	U01	Start the program.	See a clearly recognizable animation of SUBJECT.		✓	HD
T02	U05	Press 'Space' to jump	See the baker jump		✓	HD
T03	U06	Press left/right arrow keys to move in the respective direction	See the baker move horizontally		✓	HD
T04	U07	Asses the collision barriers for the baker	See the baker not leave the screen border		✓	HD
T05						
T06						
T07						
T08						
T09						
T10						

Tool 1:

Creating Class Diagrams
(https://draw.io)
We used draw.io as it has a user-friendly interface oriented around the construction of diagrams, in our case the class diagrams. These diagrams are seen in our Design and Architecture section of our report, and provide necessary information regarding the interaction/relationship of our code while remaining easy to follow.

Tool 2:

Finding Errors
(https://pmd.github.io)
Through the Zita-PMD extension, which specialises in Java code, it finds common programming flaws that may otherwise be overlooked. Within our code, it found several errors concerning variable length to event handling rules, outlined in the Software Quality section of our report.

Conclusion

Our experience in developing the animation project for the I2SSE contest has been marked by perseverance, collaboration, and growth. From the inception of the project, where we outlined our vision and objectives, to the final delivery of an engaging and interactive animation.

Throughout the development process, our team embraced a multifaceted approach, with each member contributing their expertise and efforts to different aspects of the project. From defining product requirements and managing project timelines to designing animations, coding functionalities, and documenting our progress, every facet of our project received careful attention and meticulous planning.

As we reflect on our project, it's evident that our success was not merely the result of individual efforts but rather a steady process of teamwork and collaboration. Despite encountering challenges along the way, we remained resilient and adaptive, leveraging feedback and lessons learned to refine our approach and deliver a product that exceeded expectations.

Our project, featuring a dynamic interaction between a baker and a trailing tomato within a kitchen environment, showcases our collective creativity, technical skills, and collaborative process. By adhering to an object-oriented programming approach and maintaining simplicity in our design, we ensured clarity, modularity, and ease of maintenance for our codebase.

Our efforts were recognised as we were honoured as the semi-finalist winners of the I2SSE contest for Class (C05), a reflection to our team's dedication, innovation, and ability to overcome challenges. From the initial stages where our performance in the interim report fell short of expectations to the final delivery of a successful project, our experience exemplifies the importance of resilience, adaptability, and teamwork in achieving shared goals.
Upon completing our project, we acknowledge we have accomplished together and look forward to future opportunities to collaborate, innovate, continue to progress our skills in software engineering and animation.