San Francisco State University
SW Engineering CSC 648/848

Milestone 4

December 04, 2024

Section 04 — Team 02

Members

Katy Lam	**Team Lead**
Arjun Singh Gill	**Back-end**
Matthew Aaron Weesner	**Back-end**
Niko Galedo	**Front-end**
Kevin Lam	**Front-end**
Kullathon “Mos” Sitthisarnwattanachai	**Git Master**
Arizza Cristobal	**Scrum Master**

Revision History

Version	Date	Summary
1.0	2024-12-04	First version submitted for instructor approval
0.2	2024-11-20	Updated information
0.1	2024-11-13	Initial draft proposed to Team Lead

QA Testing

Unit Testing

P1 Features

End to end tests are located at: /application/frontend/tests

Back end tests are located at: /application/backend/app/tests

Login/Logout

Description: Users can do basic user authentication functionalities such as logging in and logging out.

Description of Functional and Statement Coverage:
The tests checks that users can successfully log in with correct credentials and log out without issues. It also checks that every line of code related to these authentication actions is executed during testing, ensuring that all functions are fully tested, including error handling for invalid credentials and the logout process.

Manually Mine Resorurce

Description: Users can manually mine resources by clicking or selecting options.

Description of Functional and Statement Coverage:
The tests check if the canvas loads correctly, if sprites (e.g., humans, cars) are interactive and clickable on the map, and if the inventory can be opened. They also verify if individual spawn points for sprites are clickable and whether sprites, once clicked, are correctly stored in the inventory.

The unit test simulates the clicking of the user, sprites (objects) being clicked on and stored into the inventory. Using phaser, the canvas has everything set up including where the sprites are spawned, how they move, and UI of different menus. We use screenshots of before and after to determine if the action was effective and passing the test.

Automated Mine Resource

Description: Users can have an automated collection of resources using in-game miners.

Description of Functional and Statement Coverage:
The tests check if the game canvas loads correctly, if the UI for building works, and if the button for building a miner opens the miner placement scene. They also verify that clicking on different resources to mine matches the corresponding icon, that resources are correctly

incremented in the inventory, and that the miner is added to the miner array.

This simulates the user building the auto miners and making sure that the miners are running correctly with what is being mined, and the inventory increments correctly with the resources.

Manually Craft

Description: Users can craft items manually by selecting recipes and resources.

Description of Functional and Statement Coverage:
The tests check if the game verifies if the recipe selection works, if resources are correctly added to the crafting process, and if the item is successfully created.

Automated Craft: Crafter

Description: Users can automatically craft with a single input to output conversion per tick.

Description of Functional and Statement Coverage:
The tests check if the canvas is loaded, the game menu button for building works, and if the build menu shows up when clicked. They also verify if the crafter placement scene appears when selecting to build a crafter, if crafting recipes cycle correctly in the inventory, if the build button creates a crafter in the array, and if inventory items are crafted correctly based on the recipe.

Integration Testing

P1 Features

Integration Tests are being ran automatically using playwright. For more information see this workflow.

Login/Logout

Test cases:
The test cases verify that users can log in with valid credentials, ensuring they are granted access to their account. They also check if users can log out successfully, ensuring their session ends correctly. Additionally, tests check for proper handling of invalid login attempts and the redirection behavior after logging in or out.

Test results:
The login process works successfully with valid credentials, and users are able to log out without issues. Invalid login attempts are properly rejected, and the user is redirected to the login page. The logout process correctly ends the session, returning the user to the homepage or login page.

Analysis:

These test cases confirm that the basic login and logout functionality is working as expected. The tests verify that users can access their accounts with valid credentials, and they can safely log out, ensuring secure session management. The handling of invalid login attempts ensures the system properly restricts unauthorized access, and the redirection behavior is consistent across all tested scenarios.

Manually Mine Resorurce

Test cases:
The test cases covered five key functionalities. The first test checked if the webpage connects successfully, ensuring the server setup is correct. The second verified the canvas display, confirming its visibility and correct location using its file ID. The third focused on detecting humans or pink cars on the map by clicking on spawn locations and comparing screenshots taken before and after interactions. The fourth case tested the inventory button's functionality, ensuring it could open and close the inventory as expected, with screenshots validating the changes. Lastly, the fifth test verified that the inventory correctly updated after interactions with spawn locations and confirmed the updates through screenshot comparisons.

Test results: The webpage connected successfully, demonstrating that the server was correctly configured. The canvas was visible and properly located on the webpage. Object detection tests for humans and pink cars were mostly successful, but occasional timeouts or missed interactions occurred. Inventory interaction worked as intended, with screenshots confirming the inventory could be opened and closed seamlessly. For dynamic updates, the inventory correctly reflected changes; however, the clicker occasionally failed to target moving objects accurately, causing some tests to fail.

Analysis: The connectivity test ensured the local environment was set up properly and the webpage was accessible. The canvas display verification confirmed that the layout rendered correctly and the file ID was linked appropriately. Object detection tests validated interaction capabilities, though occasional timeouts and missed spawns indicated the need for optimization to improve reliability. Inventory interaction tests confirmed seamless operation, with UI responsiveness validated by before-and-after screenshots. Dynamic updates were generally accurate, but the clicker’s difficulty in tracking moving objects like humans or pink cars revealed the need for enhanced tracking logic to reduce errors and improve consistency.

Automated Mine Resource

Test Cases:
This test begins by loading the game canvas and verifying that the UI elements are present. It will simulate a player clicking the button to build a miner, which opens the miner placement scene. The test will then select a resource node and place the miner. After placement, the test will confirm that the miner has been added to the miner array and that the selected resource is incrementing in the inventory over time. Finally, the inventory window will be opened to confirm the updated resource count.

Test Results:

The test successfully loads the game canvas, displays the UI for building a miner, and opens the miner placement scene when the button is clicked. Once a miner is placed, it begins automated resource collection, and the selected resource increments in the inventory. The miner is correctly added to the miner array, and the updated inventory accurately reflects the resources collected. Results may vary depending on the number of resources generated during the test, but the overall functionality is confirmed to work as expected.

Analysis:
This test runs reliably under normal conditions but may be influenced by factors such as the tester's machine speed or latency in accessing the game environment. In cases of high load or unexpected delays, the miner placement or resource incrementing process might appear slower. However, these occurrences have not been observed during testing.

Manually Craft

Test cases: This test will click in locations where human sprites will appear. After a number of clicks the test will open the inventory window to show updated inventory. It will then close the inventory window and open the crafting window. It will click the craft button and open the inventory to confirm the manual craft has been completed.

Test results: the results may vary in terms of how many resources are collected, but currently the results are observed to be positive. A human sprite resource is collected. The proper windows open and the crafting is completed.

Analysis: Depending on the testers machine speed and access to the web page or some other unknown variables, there may be instances where the test is ran too early or too late which could alter the results. We have yet to see this be true, but could be an issue in the future. Otherwise the test is working as it should and producing the outcome we would like.

Automated Craft: Crafter

Test Cases:
This test starts by verifying that the game canvas is loaded and all relevant UI elements are functional. It will simulate clicking the game menu button for building, ensuring the build menu appears. Next, the test will confirm that clicking the crafter icon opens the crafter placement scene. A crafter will then be placed, and the test will verify that crafting recipes cycle correctly using the recipe index in the inventory. The test will check if the build button successfully adds a crafter to the crafter array and if the crafting process accurately consumes inventory items and produces the correct outputs as per the recipe.

Test Results:
The test successfully confirms that the game canvas is loaded, and the game menu button for building operates as expected. Clicking the crafter icon opens the crafter placement scene, and placing a crafter correctly adds it to the crafter array. The crafting recipes cycle properly based on the recipe index, and items in the inventory are consumed and crafted accurately into the recipe output. The inventory reflects the correct changes after crafting.

Analysis:
The test performs as expected under normal conditions, verifying that the crafting system is operational and integrates seamlessly with other game elements. Potential variability in results could arise from delays in accessing the build menu or recipe cycling due to system speed or network issues. However, these issues have not been observed during testing.

Coding Practices

Coding Style

For our project, we use an opinionated formatter to maintain a consistent coding style across all components. On the frontend, we utilize Biome, while for the backend, we rely on Ruff. The coding style is enforced through pre-commit hooks that automatically check code before it is committed. Additionally, our continuous integration (CI) pipeline runs linting tests on pull requests (PRs), ensuring that the code adheres to the established standards. Pull requests are only accepted if they pass the linting checks, guaranteeing high-quality and uniform code throughout the project.

CI Workflows

These workflows are run on pull requests to enforce coding style.

- /actions/workflows/lint-frontend.yml
- /actions/workflows/lint-backend.yml

Relevant Source Files for P1 Features

Login/Logout

- /application/frontend/src/routes/login.tsx
- /application/backend/app/api/routes/login.py

Manually Mine Resorurce

- /application/frontend/src/game/scenes/ui/build/buildMenu.ts
- /application/backend/app/api/routes/resources.py

Automated Mine Resource

- /application/frontend/src/game/scenes/ui/build/minerPlacement.ts
- /application/backend/app/api/routes/items.py
- /application/backend/app/api/routes/facilities.py

Manually Craft

- /application/frontend/src/game/scenes/ui/build/crafterPlacement.ts
- /application/backend/app/api/routes/items.py
- /application/backend/app/api/routes/resources.py

Automated Craft: Crafter

- /application/frontend/src/game/scenes/ui/build/crafterPlacement.ts
- /application/backend/app/api/routes/resources.py

API Documentation

All P1 features utilize the APIs created in the backend. Whenever the API is updated, the documentation is automatically generated using Swagger UI, ensuring it remains up-to-date. The latest changes merged into the master branch are automatically deployed to the staging server, along with the updated API documentation (api.staging.guacamolerigatoni.com/docs). Additionally, test coverage documentation is generated using Pytest and uploaded via Smokeshow, providing a clear view of the backend's test coverage. For example, the latest coverage report can be found at: https://smokeshow.helpmanual.io/6q6r292a3x6p6n2r146z/.

Team

Members

The list of team member names and their roles are repeated here:

Member	Role
Katy Lam	Team Lead
Arjun Singh Gill	Back-end
Matthew Aaron Weesner	Back-end
Niko Galedo	Front-end
Kevin Lam	Front-end
Kullathon “Mos” Sitthisarnwattanachai	Git Master
Arizza Cristobal	Scrum Master

M4 Checklist

The following checklist have the submission requirements for Milestone 4.

Item	Status
1. **Unit Test** ☑ Description of 5 P1 features. ☑ GitHub URL of the test cases directory. ☑ Description of functional and statement coverage. ☑ (Optional) Description of CI workflow integration.	DONE
2. **Integration Test** ☑ Description of test cases and results for 10+/all P1 features. ☑ (Optional) URL to the bug tracker system recording failed tests. ☑ Analysis of test coverage.	DONE
3. **A Coding Style** ☑ Select and describe your coding style (e.g., Google Python Style). ☑ Explain enforcement method (tool or plugin). ☑ **Source Files:** List files demonstrating adherence to coding style.	DONE
4. **Documentation of Generation [Extra Credit]** ☑ Add comments for public methods/classes in source files. ☑ Generate HTML documentation from comments.	DONE
5. **REVIEW Unit Test with Team** ☑ Are test cases covering units for the 5 P1 features? ☑ Functional Coverage >= 90%? ☑ Statement Coverage >= 50%?	DONE

- ☑ CI integration implemented (Extra Credit)?

6. **REVIEW Integration Testing with Team** DONE
   - ☑ How many P1 features were tested?
   - ☑ How many P1 features passed integration tests?

7. **Scrum REVIEW & FINALIZE Documentation** DONE
   - ☑ Ensure all required items are documented clearly. (edit any necessary)
   - ☑ Verify submission URLs are accessible.