Software Testing Report

Cohort 1 Group 8

Roy Asiku, Tianqi Feng, Andrew Jenkins, Nicholas Lambert, Tom Byron

When creating tests for our project, we decided to use unit tests using the JUnit library. We created automated tests for most of our game's key functionality, which can all be run with the click of a button. We decided on this approach due to its ease of use, speed and traceability. When working on code for the game, having to manually check whether everything is working can be very time consuming and can often lead to potential bugs being missed. The ability to use automated tests solves both of these problems by allowing us to test the game quickly and effectively. When an issue does arise, it is also much faster and easier to locate its source, due to the unit tests providing error traces. Unit tests also allow us to "lock in" functionalities of the game, essentially determining how the logic in the game should work, which helps to prevent confusion between group members on how a certain feature should work. In order to create unit tests that could function within the LibGDX environment, we utilised an open source GdxTestRunner tool made by Tom Grill: https://github.com/TomGrill/gdx-testing

We decided to group our tests into individual classes which test for a certain aspect of the game's functionality:

Asset Tests
All tests in this class simply check for the existence of assets at path locations specified in the game's variables. In order to reduce discrepancies between the unit tests and code, we added all imported assets' path locations to its class's global variables as final and static. All of these tests have passed in the current implementation, indicating that all required assets are present.

Game Tests
This testing class creates a new instance of the game, checks for the existence of all manager objects that must be generated for the game to function and tests for map changes upon interaction with a building. In order to create a new game instance in our test environment, we utilised the Mockito library to allow us to replicate the behaviour of the game upon startup. Currently, these tests do confirm that all necessary manager objects are not null after startup, and that the map changes to the desired location upon interaction with a building.

Map Tests
To test the functionality of our maps, we created tests to validate that our map manager tool correctly locates collidable tiles, exit tiles and activity tiles; that our current map path changes when loading a new map; and that our activity tiles correctly initialise and return the desired area. We again had to utilise the Mockito library to create an instance of our map manager tool within the test environment. As of the current implementation, all of these tests succeed, indicating that our map systems are fully functional. The only drawback here is that we did not create tests for each individual map, as this would be very tedious, and would restrict future ability to edit said maps.

Player Tests
Our player test class checks that the game's day cycle functions correctly upon sleeping; that the player's study, relax and eat counts increment correctly upon performing the relevant activity; that the player can and can't perform activities when appropriate; and the game

ends after 7 days have passed. These tests are fundamental to our gameplay loop, so are crucial to run after any major code changes, thus we have ensured that all of these tests pass.

Input Tests
Input tests are again crucial to the player's ability to interact with the game. The tests in this class check whether the player state correctly updates when hitting a movement or interaction key, as well as ensuring that the player cannot pass through collidable tiles. The player's current state determines the player's movement and interaction, as of our current implementation, all of these tests pass.

Activity Tests
As we have already checked the functionality of our activities within our player tests, all that's left to check here is our in game tooltips. When the player goes to perform an activity, a dialogue box appears to inform the player of what that activity involves. This dialogue changes depending on the length of the activity and its energy cost. We wanted to test that this dialogue accurately reflects the activity, so we created tests that verify activity descriptions for all activity types. Currently, these tests all pass.

Building Tests
We had to make sure to check that the buildings within our maps had the correct parameters upon being initialised and that they functioned correctly. These tests check that a building within our game's building list initialises correctly; that the building's visibility can be toggled; that the building correctly returns its associated activity; and that the game correctly calculates whether the player is within range of interacting with a building. We initially had some trouble understanding the coordinate and range systems of the previous implementation, but upon figuring them out, we got these tests to pass.

Graphics Tests
As we have already tested the existence of required assets and the functionality of our maps, we decided to finish up testing our graphics by verifying the position and scale of our camera setup and the initialisation of our game's UI, all of which have passed. Unfortunately we are not able to simulate full UI interaction within our test environment, so we can only check for the initial states of our components.

Game Object Tests
This testing class tests the functionality of the day, time and energy classes. For our day class we test its initial state upon initialisation as well as whether study, relax and eat counters increment and return correctly. Our energy tests also test its initial state; that energy decrements correctly; that the player cannot use more energy than they have; and that energy resets correctly on a new day. Time tests check that the week starts on monday; that the day starts at 8:00am; that time can increase within its given limit; that time cannot go past midnight without cycling to the next day; and that the week completes upon reaching sunday. Although we did have some issues with this testing class as the previous implementation did not have proper time restraints, these issues were resolved and now work with our current version.

Helper Tests

Finally, we ensured to test the functionality of the game's helper classes, which include saving and loading the user's high score as well as loading and initialising animations. To get the high score tests to work, we had to revamp the saving system to utilise LibGDX's preferences system rather than the previous implementation's text file system. These tests all now pass and the preferences system was also helpful when implementing our new added requirements.