Group 21
Team 21

Testing

Doaa Doukh
Surbhi Lahoria
Sean Abong
Peter Beck
Isaac Ohara
James Cretney
Lloyd Newton

4a. Testing methods

When deciding on the testing method for our program we decided upon a mixture of automated user testing and a series of user tests. We chose this as we felt that a user test would be appropriate to encapsulate the requirements of the game in terms of gameplay, and then automated testing could be used to rigorously ensure background features function as intended. This is due to the fact that it was initially thought that a large amount of the features present in the Libgdx engine could be run headless for the purpose of unit testing, however this was not the case since Libgdx requires Opengl wrapping to handle textures, which meant that texture handling could not occur when no textures were available. So rather than a 60 :40 split between favouring automated testing, we had to adapt our strategy to place greater emphasis on user testing, so an 80:20 between user tests and automated testing. This adjustment ensures that while automated tests cover the non-graphical components such as the scoring logic and the audio handling, user tests comprehensively evaluate gameplay and graphical features, providing a balanced approach to testing in our program.

4b. Report on tests

For the testing, whilst some could be done via unit tests in continuous integration, a large amount of tests had to be done via user testing. This was due to the way Opengl handles loading textures in headless mode meaning that creating a continuous integration method for the classes requiring textures was difficult. Instead game logic that could be considered separately such as the volume sliders or event logic is implemented via unit testing, and then any other tests are feature tested in the user tests. A solution to this issue would be to decouple the texture handling from the class creation and feature logic in the form of separate classes, however this would significantly modify the structure of the program solely for testing features that are available to the user, and as such it was decided against.

For the user testing to be comprehensive, a script was followed, which involved opening the leaderboard on the start menu, starting the game, entering the classroom, selecting 1 hour and performing the minigame, entering the gym selecting 1 hour, Dining, Entering the classroom for 4 hours and performing the minigame, Sleeping, At game end check the leaderboard again.On alternating days the final event changed between sleeping and letting the day run out, in order to test if the days functioned as intended, i.e when time becomes a set hour the sleep event occurs. There were also informal tests where the game was simply played through and any unintended behaviour was noted. As a result, between releases there were very few cases of unintended behaviour visible during the run of the game, rather most of the errors came about as a result of game features being unimplemented. In particular, we struggled with the minigame implementation, so most of the testing, excluding the manual tests, were not a part of it. This led to our minigames being relatively untested, so we couldn't fully check it, leading to some bugs that we couldn't have predicted.

For the automated testing, we chose to make 2 sets of simple tests, the first verifying that the functions to increase and decrease the volume worked, since this did not involve texturing and was simple to implement as it involved using the volume change function and checking that it changed or stayed the same once the volume increased beyond a certain value. For event manager the tests involved checking that a score ws only output when events occurred, and that scores could only be greater than 0. The tests for the score streaks checked that the streak checks that multiple events in multiple days in a row counted as a streak, and that streaks do not count if a streak does not exist. We felt that this would be comprehensive enough as for scoring the specific value did not matter, rather the amount of studying

relates to the scoring, and for the streaks all that mattered was that the system could keep track of the days accurately.

C.

The coverage and test results can be found at https://github.com/Team21Eng1/Diagrams-and-gantt-charts.git, where a audiomanagerTest and EventManagerTest are the testing results and test coverage indicates test coverage
Testing done on the 13th of May

Program functionality	Intended	Actual	Requirement Affected
Implement energy levels	An energy bar is displayed on the main game menu	The energy bar is displayed in the top right corner	UR_ENERGY
When an action requiring energy is performed, energy decreases	When interacting with an event, the energy decreases	Events cost energy as required, they also cannot be performed without energy	UR_INTERACTION UR_ENERGY
You cannot perform actions with too little energy	Once the energy is at zero, either the day ends or the event cannot be performed without energy	Events cannot be performed without energy	UR_ENERGY
Track the in game time	Day count is measured and displayed in game	Day count is displayed after the end of each day	UR_DAY_NIGHT_CYCLE
If the day has ended, change the date	The day count increases after sleeping or after running out of time	Day count increments as intended	UR_DAY_NIGHT_CYCLE
After 7 days display the end screen	An end screen is displayed after 7 days showing the score	An endscreen is displayed after the game is completed	UR_GAME_OVER UR_ENDING
Ensure that the score is calculated at the end of the game	A score is displayed on the main game screen	No score is displayed	UR_ENDING
Penalise over studying	A score with over studying is less than one without over studying	Scoring is higher with more studying streaks	UR_STREAKS UR_STUDYING
Generate a leaderboard	A leaderboard can be viewed displaying the scores of every	A leaderboard is generated but no scores are displayed	UR_PLAYER_SCORE

	completed run of the game		

Testing done on the 20th of May

Program functionality	Intended	Actual	Requirement Affected
Implement energy levels	An energy bar is displayed on the main game menu	The energy bar is displayed in the top right corner	UR_ENERGY
When an action requiring energy is performed, energy decreases	When interacting with an event, the energy decreases	Events cost energy as required, they also cannot be performed without energy	UR_INTERACTION UR_ENERGY
You cannot perform actions with too little energy	Once the energy is at zero, either the day ends or the event cannot be performed without energy	Events cannot be performed without energy	UR_ENERGY
Track the in game time	Day count is measured and displayed in game	Day count is displayed after the end of each day	UR_DAY_NIGHT_CYCLE
If the day has ended, change the date	The day count increases after sleeping or after running out of time	Day count increments as intended	UR_DAY_NIGHT_CYCLE
After 7 days display the end screen	An end screen is displayed after 7 days showing the score	An endscreen is displayed after the game is completed	UR_GAME_OVER UR_ENDING
Ensure that the score is calculated at the end of the game	A score is displayed on the main game screen	No score is displayed	UR_ENDING
Penalise over studying	A score with over studying is less than one without over studying	Scoring is higher with more studying streaks	UR_STREAKS UR_STUDYING
Generate a leaderboard	A leaderboard can be viewed displaying the scores of every	A leaderboard is generated but no scores are displayed	UR_PLAYER_SCORE

	completed run of the game		
Check streak achievements	A display of the current Event streak is shown	Studystreaks affect scoring, but not much else	UR_STREAKS