Group 16 Phase 3 Report

Test quality and coverage

Features covered by unit tests:

Class tested	Features tested
Cell	● setContent() : ensures content is set (static) ● setContent() : ensures content is set (movable) ● clearCell() : ensures all content is cleared
GameBoard	● getCell() : cell with correct x and y is obtained ● setCell() : cell is set with corresponding content ● removeObject() : object properly removed from the “gobjects” array and the game board
MenuState	● startGame() : game state changed to running
RunningState	● RunningState() : currentLevel should be equal to 1 after creation ● resetLevel() : currentLevel unchanged ● nextLevel() : currentLevel increased ● setLevel() : corresponding currentMap is created and currentLevel set accordingly
DepthFirstSearch	● searchPath() : return correct path from current position to main character
ScoreBoard	● increaseHealth() : health is increased by specified value and health does not go over max_health ● checkState() : if health is <= 0, level resets
RegularReward	● collision() : the reward is removed from the board and player score increased
BonusReward	● update() : variables for “flashing” effect are updated correctly

	● collision() : the reward is removed from the board and player score and health increased respectively
EndCell	● collision() : nextLevel() is called
Punishment	● collision() : the punishment is removed from the board and player health decreased
Enemy	● Enemy_search() : action list will lead enemy to player ● update() : action list changes if player moves ● collision() : the enemy is removed from the board and player health decreased ● move() : action removed from list and new destination set
MainCharacter	● move() : new destination is picked correctly ● canMoveTo() : only returns true if cell does not contain wall ● setDestination() : destination is correctly set ● arriveAtDestination() : destination is correctly set as new home ● atDestination() : position gets closer or is equal to destination ● update() : make sure current frame is correct
Map	● makeNewMap() : the correct map is created ● makeNewElement() : the correct element is created
MapFactory	● makeMap() : the corresponding map returned

Interactions covered by integration tests:

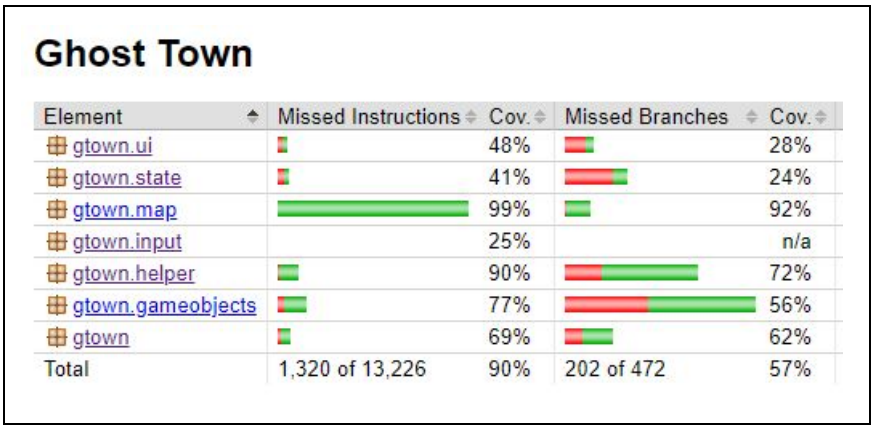
Integration Test	Interactions between
EndCellIT	Character movement, rewards collected and end cell
MovementIT	Key presses on the keyboard and character movement
MouseClickIT	Mouse clicks and buttons on the screen
PlayerAndGhostIT	The main character and enemies
SpriteLoadAndUseIT	SpriteLoader and game objects that use it

Measures taken for ensuring the quality of our test cases

- Planning ahead: Test processes should be well planned, defined, and documented. Good documentation is the tool that builds efficient communication within the software team. So, effective planning entails the creation of the quality and test plans for a project. Before we started writing our tests, we had a group meeting to discuss which class/feature to cover and not to cover. The information was documented on Google docs so team members could look up the information as needed.
- Short test cases: A test case shouldn't have too much code; otherwise it's an indication that it should be separated into multiple test cases.

Line and branch coverage *(via code-coverage-maven-jacoco)*

OVERALL:



Ghost Town

Element	Missed Instructions	Cov.	Missed Branches	Cov.
gtown.ui		48%		28%
gtown.state		41%		24%
gtown.map		99%		92%
gtown.input		25%		n/a
gtown.helper		90%		72%
gtown.gameobjects		77%		56%
gtown		69%		62%
Total	1,320 of 13,226	90%	202 of 472	57%

PACKAGES:

gtown

Element	Missed Instructions	Cov.	Missed Branches	Cov.
GameBoard		69%		61%
GameEngine		46%		50%
Cell		83%		75%
GameEngine.new TimerTask() {...}		0%		n/a
Direction		100%		n/a
Total	199 of 663	69%	18 of 48	62%

gtown.map

Element	Missed Instructions	Cov.	Missed Branches	Cov.
LevelTwo		100%		n/a
LevelZero		100%		n/a
LevelThree		100%		n/a
LevelOne		100%		n/a
MapFactory		93%		87%
Map		86%		94%
Total	32 of 8,925	99%	2 of 26	92%

gtown.state

Element	Missed Instructions	Cov.	Missed Branches	Cov.
MenuState		35%		28%
RunningState		69%		33%
WinState		8%		5%
Total	366 of 631	41%	47 of 62	24%

gtown.game -objects

Element	Missed Instructions	Cov.	Missed Branches	Cov.
Barrier		36%		n/a
BonusReward		78%		70%
EndCell		44%		50%
Enemy		81%		51%
GameObject		100%		100%
MainCharacter		71%		56%
Punishment		75%		100%
RegularReward		78%		75%
Type		100%		n/a
Total	323 of 1,410	77%	80 of 182	56%

gtown.helper

Element	Missed Instructions	Cov.	Missed Branches	Cov.
AudioPlayer		47%		66%
BreadthFirstSearch		97%		63%
DepthFirstSearch		92%		81%
SpriteLoader		98%		100%
Total	94 of 1,015	90%	35 of 126	72%

Discussion

As expected, our line and branch coverage is significantly lower in classes we did not cover; this is most prevalent in classes with long render functions (i.e., MainMenu). Additionally, in classes like Enemy and MainCharacter where a large permutation of movements and paths could be taken, we found it difficult to achieve high branch coverage.

Findings

Features and code segments that are not covered

- AudioPlayer: We have no idea as to testing the audio functionality. As such we decided to do manual testing on this feature.
- Render() methods: not tested unless logic was added in the method; our render functions all make use of the java.awt.Graphics2D class
- WinState: This is primarily a UI element with little logic to test

What we have learned from writing and running our tests

- We have learned that software testing is really necessary to point out the defects and errors that were made during the development. In the meantime, we have gained confidence on the quality of the project.
- Aside from defects and errors, we also have learned that tests are indicators of the quality of the code. For example, if it's difficult to mock a certain object inside a class method, it's better to pass that object as a parameter to the method.
- Implementation changes more frequently than an interface, so writing tests that are implementation agnostic helps us to achieve the requirement, in other words, test the state, not the implementation behavior.

Changes to the production code during the testing phase

- Switched from depth first search to breadth first search
- We reorganized some of the classes into different packages
- Changed privacy of properties to protected (i.e., MainCharacter.destination)

Fixed bugs and improvements to code

BUGS

- Fixed a bug that caused the enemies to gain a burst of speed at the beginning of the game if the player waited in the menu for a period of time before starting the game
- Fixed a bug that would cause the game to crash if the player collided with the enemy
- Fixed a bug that would prevent player-enemy collisions after the first when the player stands still
- Fixed a bug causing the reward count to persist through level change

IMPROVEMENTS

- Added a debug property to GameEngine that prevents audio from being played when in debug mode (for tests)
- Improved enemies' ability to find the player by switching from depth first search to breadth first search.