Phase 3 Report

Unit Tests

Classes:

a) Cells: DoorCell, GrassCell, and WallCell
   - Test if the cells are constructed properly with specified properties.
   - For example, WallCell and DoorCell should be solid while GrassCell is not solid after calling the constructor. But DoorCell should turn not solid once unlockDoor() is called.

b) GameHandler
   - Checking that game states are properly handled and special events in the game (ie. player losing or winning) trigger changes
   - Checking that values are being updated correctly (ie. clock)
   - Checking that all elements (ie. map board and player) needed for a game are not null after resetting the game.

c) GroundItems: Coin, Diamond, Key, and Trap.
   - Test whether goundItems correctly calls the constructor with the correct EntityID, sets the position correctly, and correctly determines the number of points gained.
   - For example, a coin should have EntityID as “Coin” and gain 5.

d) KeyControl
   - Test whether the boolean of specified keyPressed reacted after pressing the key and releasing that key.
   - Test whether the function of PressedAny returns true when at least one of the W/A/S/D keys is pressed.

e) NullEntity
   - Ensuring the null entity can be created
   - Ensuring the created entity has the correct ID: “NullEntity” returned

f) Position
   - Test whether all constructors (default, without parameter, with parameter) are setting the positions as expected.
   - Test whether both the x and y positions are returning the correct value either an int or a double.

Integration Tests

Classes:

a) Board and Cells
   - Test whether the function unblockedCell can detect if a given position on the map is unblocked, or blocked with either a wall or a door.
   - For example, the function should return true when the given position is a GrassCell and False otherwise.

b) Board and GroundItems
   - Test whether the function can detect if a given position on the map has a GroundItem on it or not.
   - Test whether the function can insert GroundItem properly into the map as expected with the given position, without putting any item on the borders, the door, overlapping another item, or inside the goal room.
   - For example, a coin should be able to be inserted into a non-boarder cell that doesn't contain any other items yet.

c) Board and Moveable
   - Test whether a movable class can move on the map board correctly with a new position given as a parameter.
   - Test whether, after a movement, the new position and original position will be updated as expected.
   - For example, after the player moves, the new position will contain this player, and the original position on the map should contain NullEntity.

d) Enemy and Board
   - Testing whether Enemy can be created and call the constructor correctly.
   - Testing whether the Enemy created is set on the correct cell on the Board
   - Testing if the Enemy created has the correct step size corresponding to the ID
   - Testing all four directions the Enemy should move in when given the corresponding player position
   - Testing trying to move when blocked, the enemy should not be able to walk through any cell that is solid.

e) Enemy, Board, and Enemy Collection
   - Testing whether the Enemy collection can set the enemies on board
   - Testing whether the board can insert enemy into the correct cell on board
   - Testing whether the Enemy Collection can update all Enemies in iteration
   - Testing whether the Enemies in the enemy collection can move when updated

f) ItemFactory, Board, and Cell
   - Testing if each item is created correctly on the map using the factory method.
   - Testing if the position on the map board and entityID are set correctly.

g) GameHandler and UI
   - Testing if each game state is calling the correct draw function and shows the expected display on the screen.
   - Testing if each resetting the game will draw a new map.

h) KeyControl and GameHandler
   - Test whether the game state changes as the P key is pressed, changing from play state to pause state or from pause state resume to play state.
   - Test whether pressing enter will correctly change the game state to the corresponding button that the user selected.

i) KeyControl and UI
   - Test if pressing W/S keys will change the selected button up and down correctly or not in the title, pause, win, and lose scene.
   - Test if pressing A/D keys will change the selected button left and right in choosing the character scene.
   - Test if pressing keys will cause the select button to be out of the index.

j) MapFactory and Board
   - Testing setting different cells on Board, asserting instance of cell

k) Player, Board, KeyControl
   - Testing moving in four directions by manually setting key conditions of KeyControl
   - Testing moving across cells, (moving from one cell array to another)
   - Testing moving in four directions when moving in that direction is blocked
   - Test moving through the door when the door is closed/open
   - Test picking up an item when moving onto a cell containing an item, the item should be removed and the player should gain points or a key
   - Testing if the player can still move when is inside the goal cell, but won't be able to move once the player touches the goal sister.

Test Quality and Coverages

Coverage result:

Element	Class, %	Method, %	Line, %
com.group5.twins_game	92% (26/28)	95% (106/111)	85% (982/1149)
Board	100% (1/1)	100% (7/7)	100% (32/32)
Cell	100% (1/1)	85% (6/7)	89% (17/19)
Coin	100% (1/1)	100% (2/2)	100% (6/6)
Diamond	100% (1/1)	100% (2/2)	100% (6/6)
Direction	100% (1/1)	100% (2/2)	100% (2/2)
DoorCell	100% (1/1)	100% (5/5)	90% (18/20)
Enemy	100% (1/1)	100% (4/4)	98% (63/64)
EnemyCollection	100% (1/1)	80% (4/5)	72% (8/11)
Entity	100% (1/1)	100% (4/4)	77% (7/9)
GameHandler	100% (1/1)	100% (8/8)	92% (84/91)
GoalCell	100% (1/1)	100% (2/2)	100% (5/5)
GrassCell	100% (1/1)	100% (2/2)	89% (17/19)
GroundItem	0% (0/1)	100% (0/0)	100% (0/0)
ImgLoader	100% (1/1)	100% (1/1)	50% (4/8)
ItemFactory	100% (1/1)	100% (4/4)	66% (16/24)
Key	100% (1/1)	100% (3/3)	100% (10/10)
KeyControl	100% (1/1)	80% (4/5)	94% (123/130)
Main	0% (0/1)	0% (0/1)	0% (0/8)
MapBuilder	100% (1/1)	100% (4/4)	100% (130/130)
MapFactory	100% (1/1)	100% (4/4)	100% (14/14)
Movable	100% (1/1)	100% (1/1)	100% (1/1)
NullEntity	100% (1/1)	100% (1/1)	100% (2/2)
Player	100% (1/1)	100% (5/5)	96% (79/82)
Positions	100% (1/1)	90% (9/10)	93% (14/15)
Score	100% (1/1)	100% (5/5)	100% (7/7)
Trap	100% (1/1)	100% (2/2)	100% (6/6)
UI	100% (1/1)	100% (12/12)	72% (304/419)
WallCell	100% (1/1)	100% (3/3)	77% (7/9)

- GroundItem class got 0% Class converges but 100% on all the others because this is an abstract class that we won't call this abstract class's constructor.
- Some of our class's line coverage is not 100% because the rest of the not covered part is the codes that catch the IOexceptions for reading images.

Findings

Our original design in the UI class of commandNum has continued to be used no matter if the state changes or not. So, we changed the implementation which sets the commandNum to one each time the game state changes. This indicates that the button will default be selected as the topmost button for every state that has buttons.

Our process of testing also helped us eliminate various statements or functions that were unnecessary. For example, under the gains() method in Entity.java, we had an if statement that would never be true, as the only time it would be true is when the method itself had been overwritten (and therefore never called). This was very obvious during testing, as it was impossible to reach full coverage. There were various other places where a small part of the code had unnecessary statements or conditions, and the testing phase allowed us to find and remove those parts.