System and Unit Test Report - "Battle for Westeros" - Wildfire Studios - 7/24/18

System Test Scenarios

Sprint 1:

- All user stories for Sprint 1 were reserved for team members to either research and get accustomed for tools or brainstorm features and mechanics to have in the game. Therefore, no system test scenarios were needed for Sprint 1.

Sprint 2:

- User Story 1: As a user, I want to see the player character on the screen with aesthetic graphics, with the capacity to change said graphics based on certain interactions because one of the high level goals is to have interactive graphics.
    - At the homepage/game menu, select "Fight Menu", then the red/green level button
    - User should see player snapped to a tile belonging to an aesthetically pleasing and technically functional map

- User Story 2: As a user, I want my player character to be able to interact with the map and its objects, such as being restricted from traveling on water/mountain tiles so that the game experience is more immersive and interactive.
    - While the combat state is running, have blue tiles denote where I can go; red tiles denote obstacles
    - Tap on adjacent or neighboring tiles to move to that space

Sprint 3:

- User Story 2 - As a user, I want a finished combat system that's implemented with the player and item classes so that I have an actual game to play.
    - At the homepage/game menu, select 'Select Loadout'
    - Select desired weapon and armor
    - Press the phone 'back' button to return to the game menu
    - Select 'Fight Menu' to proceed with the game
    - User should enter combat with the player stats combined with the item's stats

- User Story 5 - As a user, I want each level to have a larger area so that the game feels more immersive.
    - Scenario: If my character moves onto a certain tile on the edge of the map, the screen will refresh to display a different part of the map.

Unit Tests

Items Unit Test

Team Member: Khang Tran

Test file(s): testItems.java, itemAttributes.txt

File(s) tested: Weapon.java, Armor.java

Details of the module: testItems.java will act as Items.java. It will create test weapon and armor objects based on the stats specified in itemAttributes.txt (by calling *public Weapon* and *public Armor*) and will ask the user if he/she wants to see the items' stats. This is to see if item objects are created and stored correctly.

Test cases: inputs for item type, item name, attack, defense, durability, minimum level

Equivalence classes:

- public static Weapon addWeapon(String inputType, String inputName, int inputMinLvl, int inputDmg, int inputDef, int inputDur)

- public static Armor addArmor(String inputType, String inputName, int inputMinLvl, int inputDmg, int inputDef, int inputDur)

- public Armor(String inputType, String inputName, int inputMinLvl, int inputDmg, int inputDef, int inputDur)

- public Weapon(String inputType, String inputName, int inputMinLvl, int inputDmg, int inputDef, int inputDur)

- public void printStats()

Boss Unit Test

Team Member: Thomas Zhen

Test file(s): testBoss.java

File(s) tested: Boss.java

Details of the module: In the test, it will create boss object and test the methods using different input. The user should use a player to test and see if the boss object working correctly by attacking it, try to move on to the same tile as the boss before the boss dies, see if the boss be gone when it is dead, and see if the player will get hurt etc. All in all, the test will cover every methods that need to be tested.

Test cases: Try different inputs, context, player, point

- Boss object 1: health - 100 damage - 50
- Boss object 2: health - 200 damage - 50
- Boss object 3: health - 100 damage - 100

Equivalence classes:

- public RectBoss(Context c, RectPlayer player)
- public void setHealth(RectPlayer player)
- public void update(Point point)
- public void draw(Canvas c)
- public void drawHealth(Canvas c)

Tile Unit Test

Team Member: Margarita Fernandez

Test file(s): TileTester.java

File(s) tested: Tile.java, Coord.java

Details of the module: Early Tile tester. It was used while the map was first being created to ensure that the object correctly held data such as pixel coordinates on screen and row and column on map. It creates three different Tiles and prints their information. It also tests the very basic functions of Coord.java, which simply holds an x and y coordinate and provides appropriate access and function methods.

Test cases:

- Tile A:
    - Pixel coordinates: (-150, 225)
    - Map location: Row: 0, Col: 0
- Tile B:
    - Pixel coordinates: (0, 75)
    - Map location: Row: 1, Col: 2
- Tile C:
    - Pixel coordinates: (0, -75)
    - Map location: Row: 2, Col: 2

Equivalence classes:

- public Coord(int xIn, int yIn)
- public Tile(int tileLengthIn, Coord tlIn, int row, int col)
    - void toString()
    - int getRow()
    - int getCol()
    - int getTileLength()
    - Coord getPixelCoordinates()

Enemy Unit Test

Team Member: Thomas Zhen

Test file(s): testRectEnemy.java

File(s) tested: RectEnemy.java

Details of the module: In the test, it will create an enemy object and test the methods using different input. The user should use a player to test and see if the enemy object is working correctly by attacking it, see if the enemy be gone when it is dead, and see if the player will get hurt, and see if the enemy can be on the same tile before and after the enemy is dead etc.

Test cases: Try different inputs, context, player, point

- Enemy object 1: health - 100, damage - 10
- Enemy object 2: health - 200, damage - 10
- Enemy object 3: health - 100, damage - 20

Equivalence classes:

- public RectEnemy(Context c, RectPlayer player)
- public void setHealth(RectPlayer player)
- public void update()
- public void draw(Canvas c)
- public void drawHealth(Canvas c)

PlayerProj Unit Test

Team Member: Ian Feekes

Test file(s): ProjectileTestPlayer.java

File(s) tested: PlayerProj.java

Details of the module: ProjectileTestPlayer.java acts as the RectPlayer.java in that it manages the creation of several PlayerProj objects and tests for their ability to compute target vectors and update. It will then update them several times to ensure that they continue to move in a path towards the correct vector that has been computed.
This test makes sure that all projectile functions are working properly, as our projectiles do not behave as desired. It is the RectPlayer's management of the PlayerProj that is giving us projectiles with strange behavior, as this unit test confirms.

Test cases: input arguments for creating the projectile, proper computation of movement vectors, and correct movement behavior when update method is called.

-P = new PlayerProj(new Coord(1, 1), new Coord(1, 1))

This tests for computational behavior and data allocation with projectiles that have equivalent target vectors and spawning points.

-Q = new PlayerProj(new Coord(50,50), new Coord(100,0))

This tests for the behavior of a projectile fired close range that moves with both positive and negative movement vectors, which requires specifically selective usage of Math.abs() when scaling the increment vectors.

-R = new PlayerProj(new Coord (0,0), new Coord(0,300)

This tests for the outliers to make sure that projectiles behave as planned when spawned on coordinates that are on the bounds of the screen but still barely have a legal location

-P.setLoc(P.getLoc())

Checks to make sure that it functions as properly for setting a projectile's location to its current location with no changes.

-for(PlayerProj i:projectiles) i.update() i.getLoc() i.isHidden()

Makes sure that each of the projectiles is able to update its position properly and correctly validate its new location based on what the vector should be. Also verifies that each PlayerProjectile is initialized to appear on the screen and not be hidden.

Equivalence classes:

- Public PlayerProj(Coord target, Coord spawn, Context c)
- Public void resetVector(Coord target)
- Public int[] getVector()
- Public Coord getLoc()

- Public Coord setLoc()
- Public void update()
- Public void hide()
- Public void appear()
- Public boolean isHidden()
- public void printStats()

MapManager Unit Test

Team Member: Zhizhou Jiang

Test file(s): MapTest.java

File(s) tested: MapManager.java

Details of the module: MapTest.java is created for testing some features in MapManager.java. MapTest.java includes a main function. In the main, there is an object of MapManager, and I use the object call several functions. MapTest.java has the output of lists which test has been tested and if it was pass or where you need to go to fix it.
Test cases: Constructor, getTiles, tiles holder(outer ArrayList of tiles), how many Tiles in the tiles holder.

Equivalence classes:

- `public MapManager(Context c, int x, ArrayList<RectEnemy> enemiesIn)`
- **private void** populateMap()
- **public** ArrayList<Tile> getTiles()
- **public** ArrayList<ArrayList<Tile>> getTiles_holder()