Phase 3 Report

CMPT 276 Project Group 29: Andy Cheng, Martin Lau, Andy Lu and Jackson Nguyen

Features for testing

- Move the player
    - No movement into walls
    - No movement into locked doors
    - No movement into exits without a ladder
    - Player movement from input
    - Player movement happens in only 1 direction at a time
    - Player image should correspond to what direction they're facing
- Enemy pathfinding
    - No movement into walls
    - No interaction with any other object aside from the player
    - No collision with other enemies
    - Damage dealt to player when colliding
    - Normal enemies
        - Normal enemies must attempt to move towards the player if a path exists
    - Knight enemy
        - Moves only in L patterns from each position
    - Patrol enemy
        - Chases player only if an unbroken line of sight exists from each position
        - If so, moves like a normal enemy
- Collect items (in general, assert new player score == old player score + item score)
    - Collect keys
    - Collect ladder
    - Collect coin
    - Collect Sword
    - "Collect" Trap
- Opening doors
    - 1 key is consumed to open a door
    - Door becomes open, movement no longer blocked
- Escaping into holes
    - 1 ladder is used to escape into a hole
    - Player goes into the next level if there is one, or enters the win screen
- Attacking
    - Player is unable to attack again for a period of time
    - Enemies receive damage if on same space as attack
- Player damage
    - Player takes damage when hit by enemy/traps
    - Player is invincible for a period of time

    - Player loses when health <= 0 or score < 0
- Movement allowed only every ‘tick’
    - No entities allowed to move between ‘ticks’
- GameLevelSwitcher
    - If player wins level it goes to the next level
    - When player wins the game check if player won
- UI Elements
    - Hearts: player health
    - Keys: key count
    - Ladder: ladder count
    - Sword: sword count
    - Score: player score
        - displays a number matching player score

Test Quality and Coverage

In general, we tried to write tests that covered the typical cases of our classes and methods, and wrote a few tests to cover the exceptional cases. This ensures that normal gameplay can be well-tested, but if some exceptional input occurs, we will have properly accounted for that too.

- EntityFactoryTest
    - Testing createAndSetEnemy with createEnemyTestNormal, createEnemyInvalidCoordinates
        - Both tests give line coverage = 4/4 = 100%
        - Both tests give branch coverage = 2/2 = 100%

    - Testing createAndSetPlayer with createTwoEntitiesOnAPoint
        - createTwoEntitiesOnAPoint has line coverage = 3/4 = 75%
        - createTwoEntitiesOnAPoint has branch coverage = 1/2 = 50%

    - Testing createAndSetKnight with createDifferentEnemies
        - createAndSetKnight has 4 lines, createDifferentEnemies tests 3 lines so line coverage = 3/4 = 75%
        - createAndSetKnight has 2 branches, createDifferentEnemies tests 1 branch. Branch coverage = 1/2 = 50%

    - Same thing as above with createAndSetPatrol, createAndSetSword, createAndSetKey, createAndSetLadder, createAndSetCoin (line coverage = 3/4 = 75%, branch coverage = 1/2 = 50%) [don’t test null case]

- GameInfoTest
    - Default GameInfo constructor has 100% line, branch coverage
    - Gameboard GameInfo constructor has 100% line, branch coverage
    - Gameboard + EntityList GameInfo constructor has 100% line, branch coverage

- Rows and Cols GameInfo constructor has 100% line, branch coverage
- Rows and Cols + EntityList GameInfo constructor has 100% line, branch coverage

- CheckBoardBounds
    - Has 100% line coverage (one line)
    - Has 100% branch coverage (covers both true & false cases)

- HasEntity
    - Has 100% line coverage (one line), 100% branch coverage (no branch)

- HasEntityType
    - Has 4/4 = 100% line coverage from hasEntityTypeTestWithType, hasEntityTypeTestWithoutType
    - Has 2/2 = 100% branch coverage from hasEntityTypeTestWithType, hasEntityTypeTestWithoutType

- addEntity
    - addEntityTestNormal tests 2/2 lines, so 100% line coverage
    - addEntityTestNormal and addEntityTestNull tests null/not null branches so 100% branch coverage

- getEntityAt
    - getEntityAtTestNormal and getEntityAtTestExceptional test 3/3 lines, so 100% line coverage
    - getEntityAtTestNormal and getEntityAtTestExceptional test checkBoardBounds(x, y)'s true/false cases so 100% branch coverage

- getAllListEntitiesAt
    - getAllListEntitiesAtNormal tests all 6 lines, 100% line coverage
    - getAllListEntitiesAtNormal, getAllListEntitiesAtExceptional tests 3 branches, 3/4 = 75% branch coverage

- setWallAt
    - setWallAtTests cover 2/2 lines = 100% line coverage
    - setWallAtTests cover 1/2 branches = 50% branch coverage

- findClosestReachablePoint
    - findClosestReachablePointTest covers all lines and branches = 100% line/branch coverage

- isPointReachable
    - All isPointReachable tests give line coverage = 10/10 = 100%

        - All isPointReachable tests cover all if statements and switch cases, so branch coverage is 10/10 = 100%

- PlayerDamageTest
    - Has 12/12 = 100% line coverage from EntityStateHandler
        - hitEnemyNormal tests 10 lines
        - killEnemyWithSword tests 12 lines
        - So line coverage is 12/12 = 100%
    - 8 branches of interest to test from EntityStateHandler (if statements on line 50, 52, 54, 57)
        - hitEnemyNormal tests 3 branches
        - hitEnemyWithNoSword tests 1 branch
        - killEnemyWithSword tests 5 branches
        - changeDirectionAndAttack tests 6 branches
        - outOfBoundsSwordTest tests 3 branches
        - Line 54 is not checked for false. This could cause a problem if the program tries to subtract health from a non-enemy entity.
            - So line coverage is 7/8 = 87.5%.

- PathfinderTest
    - getNextMoveTo()
      2 major branches, 5 inner branches
        - normalStartAtTarget covers 1st major branch at line 196
        - normalReachablePath covers the 5 inner branches and covers true outcome of 2nd major branch at line 224
        - normalStraightWallOneOpening test of true outcome 2nd branch
        - normalNoPathWalls Covers false outcome of 2nd major branch
        - normalNoPathDoors coverage of false outcome 2nd branch

      All 7 branches covered, 100% branch coverage

    - getNextMoveKnightTo()
      2 major branches, 9 inner branches
        - knightStartAtTarget() covers 1st major branch
        - knightReachablePath covers 9 inner branches and true outcome of 2nd major branch at line 342
        - knightNoPathWalls covers false outcome of 2nd major branch at line 342
        - knightNoPathDoors Additional coverage of false outcome 2nd branch

      All 11 branches covered, 100% branch coverage

    - getNextMovePatrolToTest
      Covers same essential logic of getNextMovePatrolTo, but does not cover Patrol
      1 major branch, 2 inner branches
        - patrolStartAtTarget Covers true outcome of 1st major branch at line 441

        - patrolReachablePathGoingRight, patrolReachablePathGoingLeft
            - Both additional testing to true outcome 1st branch
        - patrolNoPath
            - Covers false outcome of 1st major branch
            - Covers both outcomes of 2nd inner branch
        - patrolStraightWallOneOpeningNoPath
            - Additional testing false outcome 1st major branch
        - patrolStraightWallOneOpeningGoingRight covers false outcome of 1st inner branch and both outcomes of 2nd inner branch
        - patrolStraightWallOneOpeningGoingLeft covers true outcome of 1st inner branch and both outcomes of 2nd inner branch at line 451

    All 3 branches covered, 100% branch coverage

- PlayerMoveTest, basically tests EntityStateHandler
  Ignoring PlayerAttack coverage, 7 major branches, 7 inner branches
    - playerMoveNoObstacle covers default outcome of the switch, the 2nd major branch
    - playerMoveIntoWall covers the 1st major branch
    - playerMoveIntoClosedDoorNoKey
        - Covers closedDoor outcome of the switch, 5th major branch, line 102
            - Covers false outcome of the inner branch (0.5)
    - playerMoveIntoClosedDoorWithKey
        - Covers closedDoor outcome of the switch
            - Covers true outcome of the inner branch (1)
    - playerMoveIntoKey
        - Covers reward outcome of the switch, 3rd major branch, line 71
            - Covers the key outcome of the inner switch (2)
    - playerMoveIntoCoin
        - Covers reward outcome of the switch
            - Covers the coin outcome of the inner switch (3)
    - playerMoveIntoLadder
        - Covers reward outcome of the switch
            - Covers the ladder outcome of the inner switch (4)
    - playerMoveIntoExitNoLadder
        - Covers exit outcome of the switch, 6th major branch, line 117
            - Covers the false outcome of the inner branch (4.5)
    - playerMoveIntoExitWithLadder
        - Covers exit outcome of the switch
            - Covers the true outcome of the inner branch (5)
    - playerMoveIntoSword
        - Covers reward outcome of the switch
            - Covers the sword outcome of the inner switch (6)
    - playerMoveIntoTrap
        - Covers exit outcome of the switch, 4th major branch, line 88

            - Covers the true outcome of the inner branch (6.5)
    - playerMoveIntoEnemy
        - Covers exit outcome of the switch, 7th major branch, line 131

All 7 major branches covered, 6.5 inner branches covered, 13.5 / 14 = 96% coverage

Typically, the branches that we didn't cover are exceptional branches, for example, if the player is null then return nothing. We didn't test those branches since they are unlikely to occur in a real run of the game; when we create the levels, entities created are unlikely to be null if we properly added them. Also, we didn't test some switch/cases as they lead to almost identical results, just with a different entity type attribute passed in. Finally, for testing the game logic itself, there are just too many combinations of events that can happen for us to test them all. Combinations of positions, which enemies are on the board, player health, sword, keys, etc. create too many parameters, so we just tested the simple cases. In the same vein, we couldn't test display logic or user input logic at all, since those are not easily testable through code.

2.1.3 Findings

- The player lacked a way to move or attack without requiring input. Methods have been added for this to ease testing.
    - (see junit-testing-player-move)
- The Player's default invincibility ticks are 30, and the player is invincible if ticks <= 30. Only update() added ticks, so the Player would be invincible during tests without update(). The default value is now 31.
    - (see junit-testing-player-move)
- Coin's score multiplier variable used to be static. For greater consistency with other member variables, it is no longer static and now has a getter.
    - (see junit-testing-player-move)
- Testing EntityFactory (when creating enemies with invalid/out-of-bound coords) made us realize that in the isPointReachable() method of gameInfo, a null check must be done for getEntityAt(x, y).getEntityType() != Entity.EntityType.floor.
- Testing GameInfo showed that we had to change the order of rows and cols in the MazeGenerator constructor so that the number of elements in the internal game board array remains consistent.
- Also, testing GameInfo made us realize we need null checks for addEntity.
- Realized that we need to have check if the player has a sword in EntityStateHandler
- The Player was able to move into the same space as an Exit without a Ladder. This has been fixed.
- The Player's movement previously didn't check for bounds. This has been fixed.
- Pathfinder did not check for entities before moving into a space. Fixed