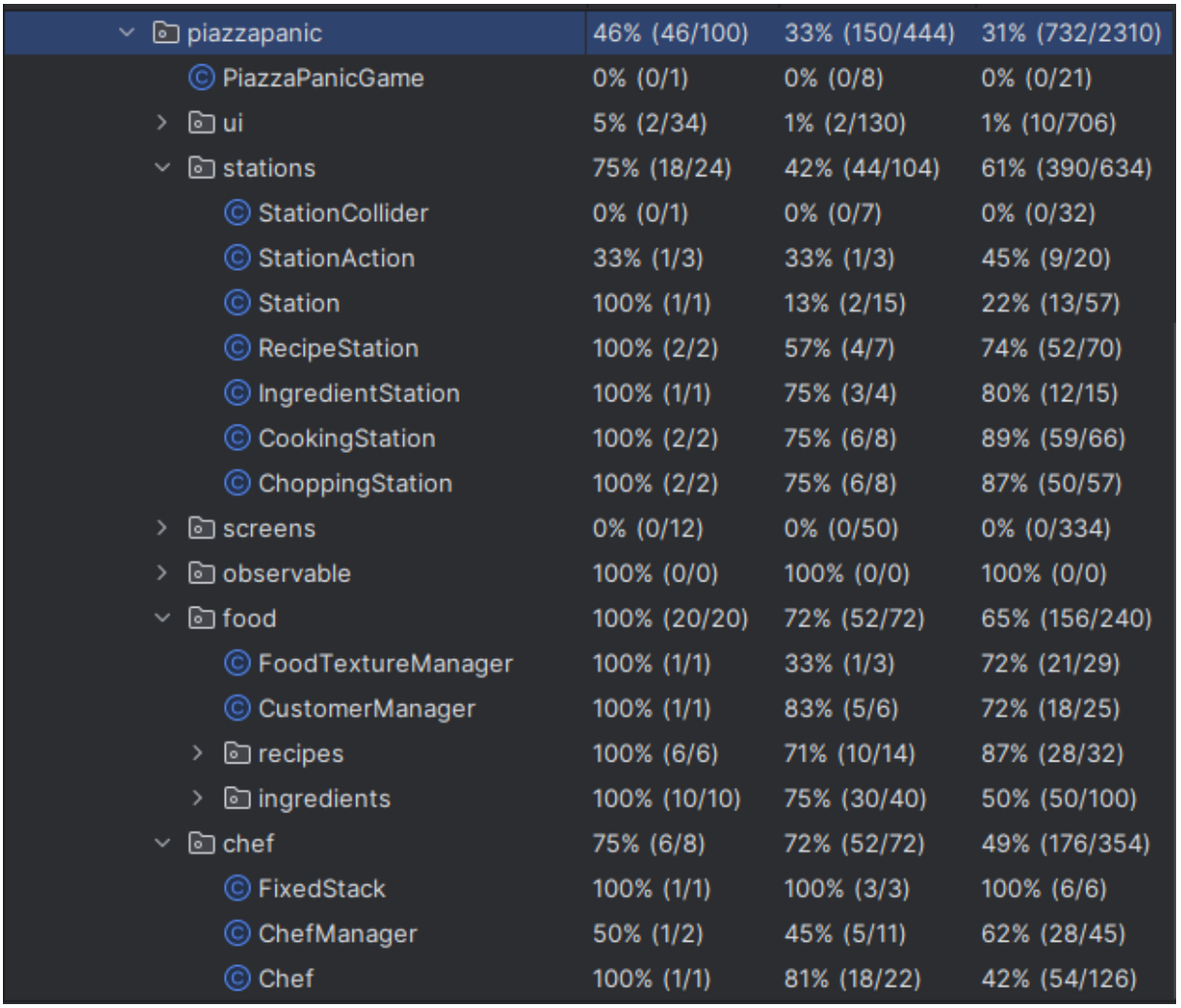
AUTOMATED TESTS:
First column: Percentage of Classes covered by tests
Second column: Percentage of Methods covered by tests
Third column: Percentage of Lines covered by tests (including branches)



piazzapanic	46% (46/100)	33% (150/444)	31% (732/2310)
PiazzaPanicGame	0% (0/1)	0% (0/8)	0% (0/21)
ui	5% (2/34)	1% (2/130)	1% (10/706)
stations	75% (18/24)	42% (44/104)	61% (390/634)
StationCollider	0% (0/1)	0% (0/7)	0% (0/32)
StationAction	33% (1/3)	33% (1/3)	45% (9/20)
Station	100% (1/1)	13% (2/15)	22% (13/57)
RecipeStation	100% (2/2)	57% (4/7)	74% (52/70)
IngredientStation	100% (1/1)	75% (3/4)	80% (12/15)
CookingStation	100% (2/2)	75% (6/8)	89% (59/66)
ChoppingStation	100% (2/2)	75% (6/8)	87% (50/57)
screens	0% (0/12)	0% (0/50)	0% (0/334)
observable	100% (0/0)	100% (0/0)	100% (0/0)
food	100% (20/20)	72% (52/72)	65% (156/240)
FoodTextureManager	100% (1/1)	33% (1/3)	72% (21/29)
CustomerManager	100% (1/1)	83% (5/6)	72% (18/25)
recipes	100% (6/6)	71% (10/14)	87% (28/32)
ingredients	100% (10/10)	75% (30/40)	50% (50/100)
chef	75% (6/8)	72% (52/72)	49% (176/354)
FixedStack	100% (1/1)	100% (3/3)	100% (6/6)
ChefManager	50% (1/2)	45% (5/11)	62% (28/45)
Chef	100% (1/1)	81% (18/22)	42% (54/126)

Total number of tests: 86
Results before Assessment 2 changes: 86 passed, 0 failed
Results after Assessment 2 changes: N/A

Testing was not completed before the implementation for assessment 2 thus not leaving enough time for the team to update and adapt the tests to the new implementation which causes some to break the compiler due to the new changes. The team therefore does not have sufficient information on how many tests pass, and/or are still valid.

MANUAL TESTS:

```
private float adjustHorizontalMovementForCollision(float xMovement) {
  if (xMovement > 0.0001f) {
    // Check right side of chef at top, middle and bottom
    float rightBorder = getX() + getWidth() + xMovement;
    Rectangle hitBoundsBottom = getCollisionObjectBoundaries(rightBorder, getY());
    Rectangle hitBoundsMiddle = getCollisionObjectBoundaries(rightBorder,
        y: getY() + getHeight() / 2f);
    Rectangle hitBoundsTop = getCollisionObjectBoundaries(rightBorder, y: getY() + getHeight());

    // Calculate new change in x relative to the collision object boundaries
    float adjustment = -getWidth() - collisionSkin - getX();
    if (hitBoundsBottom != null) {
      xMovement = hitBoundsBottom.x + adjustment;
    }
    if (hitBoundsMiddle != null) {
      xMovement = Math.min(xMovement, hitBoundsMiddle.x + adjustment);
    }
    if (hitBoundsTop != null) {
      xMovement = Math.min(xMovement, hitBoundsTop.x + adjustment);
    }
  } else if (xMovement < -0.0001f) {
    // Check left side of chef at top, middle and bottom
    float leftBorder = getX() + xMovement;
    Rectangle hitBoundsBottom = getCollisionObjectBoundaries(leftBorder, getY());
    Rectangle hitBoundsMiddle = getCollisionObjectBoundaries(leftBorder,
        y: getY() + getHeight() / 2f);
    Rectangle hitBoundsTop = getCollisionObjectBoundaries(leftBorder, y: getY() + getHeight());

    // Calculate new change in x relative to the collision object boundaries
    float adjustment = collisionSkin - getX();
    if (hitBoundsBottom != null) {
      xMovement = hitBoundsBottom.x + hitBoundsBottom.width + adjustment;
    }
    if (hitBoundsMiddle != null) {
      xMovement = Math.max(xMovement, hitBoundsMiddle.x + hitBoundsMiddle.width + adjustment);
    }
    if (hitBoundsTop != null) {
      xMovement = Math.max(xMovement, hitBoundsTop.x + hitBoundsTop.width + adjustment);
    }
  }
  return xMovement;
}
```

1)
Title: testChefHorizontalMovementAdjustmentTopRight

Description: This test checks if there is a positive movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile north-east of him, taking into account the chef's height, and adjusting the chef's movement (stopping him from crossing into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 1 and between y = 1 and y =2, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement above 0.001f.
   xMovement = 1f.
2. A calculation that finds the movement of the chef from his right side in the positive x axis.
   rightBorder = 1 + 1 + 1f
3. Checks the cells south-east, directly east and north-east of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = null, hitBoundsMiddle = null, hitBoundsTop = (2, 2, 1, 1)
4. A sum of the negative width of the chef, minus the collision skin, which is a value of 0.01f, as well as subtracting the x coordinate of the chef is made.
   adjustment = -(1) - (0.01f) - (1) = -1 - 0.01f - 1
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the “(hitBoundsTop != null)” is hit and the adjustment to movement is carried out.
   xMovement = The minimum between 1f and (2 + (-1 - 0.01f - 1) = -0.01f), which in this case is -0.01f

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = -0.01f

Result: Pass

2)
Title: testChefHorizontalMovementAdjustmentMiddleRight

Description: This test checks if there is a positive movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile directly east of him, taking into account the chef’s height, and adjusting the chef’s movement (stopping him from crossing

into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 1 and y = 2, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement above 0.001f.
   xMovement = 1f.
2. A calculation that finds the movement of the chef from his right side in the positive x axis.
   rightBorder = 1 + 1 + 1f
3. Checks the cells south-east, directly east and north-east of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = null, hitBoundsMiddle = (2, 2, 1, 1), hitBoundsTop = null
4. A sum of the negative width of the chef, minus the collision skin, which is a value of 0.01f, as well as subtracting the x coordinate of the chef is made.
   adjustment = -(1) - (0.01f) - (1) = -1 - 0.01f - 1
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the “(hitBoundsMiddle != null)” is hit and the adjustment to movement is carried out.
   xMovement = The minimum between 1f and (2 + (-1 - 0.01f - 1) = -0.01f), which in this case is -0.01f

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = -0.01f

Result: Pass

3)

Title: testChefHorizontalMovementAdjustmentBottomRight

Description: This test checks if there is a positive movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile south-east of him, taking into

account the chef's height, and adjusting the chef's movement (stopping him from crossing into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 1 and between y = 2 and y =3, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement above 0.001f.
   xMovement = 1f.
2. A calculation that finds the movement of the chef from his right side in the positive x axis.
   rightBorder = 1 + 1 + 1f
3. Checks the cells south-east, directly east and north-east of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = (2, 2, 1, 1), hitBoundsMiddle = null, hitBoundsTop = null
4. A sum of the negative width of the chef, minus the collision skin, which is a value of 0.01f, as well as subtracting the x coordinate of the chef is made.
   adjustment = -(1) - (0.01f) - (1) = -1 - 0.01f - 1
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the "(hitBoundsBottom != null)" is hit and the adjustment to movement is carried out.
   xMovement = (2 + (-1 - 0.01f - 1) = -0.01f), which in this case is -0.01f

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = -0.01f

Result: Pass

4)

Title: testChefHorizontalMovementAdjustmentTopLeft

Description:This test checks if there is a negative movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile north-west of him, taking into

account the chef's height, and adjusting the chef's movement (stopping him from crossing into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 3 and between y = 1 and y =2, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement below -0.001f.
   xMovement = -1f.
2. A calculation that finds the movement of the chef from his left side in the negative x axis.
   leftBorder = 1 + (-1f)
3. Checks the cells south-west, directly west and north-west of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = null, hitBoundsMiddle = null, hitBoundsTop = (2, 2, 1, 1)
4. A subtraction is made from the collision skin which is a set value of 0.01f with the x coordinate of the chef
   adjustment = 0.01f - 3
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the "(hitBoundsTop != null)" is hit and the adjustment to movement is carried out.
   xMovement = The maximum between -1f and (2 + 1 + (0.01f - 3)), which in this case is (2 + 1 - 3 + 0.01f = 0.01f)

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = 0.01f

Result: Pass

5)

Title: testChefHorizontalMovementAdjustmentMiddleLeft

Description: This test checks if there is a negative movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile directly west of him, taking into account the chef's height, and adjusting the chef's movement (stopping him from crossing into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 3 and y = 2, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement below -0.001f.
   xMovement = -1f.
2. A calculation that finds the movement of the chef from his left side in the negative x axis.
   leftBorder = 1 + (-1f)
3. Checks the cells south-west, directly west and north-west of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = null, hitBoundsMiddle = (2, 2, 1, 1), hitBoundsTop = null
4. A subtraction is made from the collision skin which is a set value of 0.01f with the x coordinate of the chef
   adjustment = 0.01f - 3
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the "(hitBoundsMiddle != null)" is hit and the adjustment to movement is carried out.
   xMovement = The maximum between -1f and (2 + 1 + (0.01f - 3)), which in this case is (2 + 1 - 3 + 0.01f = 0.01f)

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = 0.01f

Result: Pass

6)

Title: testChefHorizontalMovementAdjustmentBottomLeft

Description:This test checks if there is a negative movement input in the x axis for the chef, comparing the tile the chef is currently occupying with the tile south-west of him, taking into account the chef's height, and adjusting the chef's movement (stopping him from crossing into any of those tiles) if there are objects/counters the chef shouldn't be able to walk through, otherwise the chef continues moving as per usual.

Related Components: Chef

Related Requirements: FR_MOVE_PLAYABLE_CHARACTER

Authors: Galin 28/04/2023

Setup:

- 5x5 tiled map with boundaries from coordinates (0,0) and (4, 4)
- A chef at position x = 3 and between y = 2 and y = 3, of height 1 and width 1
- An ingredient station positioned at (2, 2) with width of 1 and height of 1

Steps:

1. An input of xMovement below -0.001f.
   xMovement = -1f.
2. A calculation that finds the movement of the chef from his left side in the negative x axis.
   leftBorder = 1 + (-1f)
3. Checks the cells south-west, directly west and north-west of the chef and returns either a null, or a rectangle with the coordinates and width/height of the chef/station/counter in that cell.
   hitBoundsBottom = (2, 2, 1, 1), hitBoundsMiddle = null, hitBoundsTop = null
4. A subtraction is made from the collision skin which is a set value of 0.01f with the x coordinate of the chef
   adjustment = 0.01f - 3
5. All branches are cleared and if the values for the hitBounds rectangles are null, the branch is skipped. In this case the branch only the "(hitBoundsMiddle != null)" is hit and the adjustment to movement is carried out.
   xMovement = (2 + 1 + (0.01f - 3)), which in this case is (2 + 1 - 3 + 0.01f = 0.01f)

Expected Outcome: Chef is pushed the opposite direction the value of the collisionSkin (0.01f).

Actual Outcome: xMovement = 0.01f

Result: Pass

```
private float adjustVerticalMovementForCollision(float yMovement) {
  if (yMovement > 0.0001f) {
    // Check top side of chef at left, middle and right
    float topBorder = getY() + getHeight() + yMovement;
    Rectangle hitBoundsLeft = getCollisionObjectBoundaries(getX(), topBorder);
    Rectangle hitBoundsMiddle = getCollisionObjectBoundaries( x: getX() + getWidth() / 2f, topBorder);
    Rectangle hitBoundsRight = getCollisionObjectBoundaries( x: getX() + getWidth(), topBorder);

    // Calculate new change in y relative to the collision object boundaries
    float adjustment = -getHeight() - collisionSkin - getY();
    if (hitBoundsLeft != null) {
      yMovement = Math.min(yMovement, hitBoundsLeft.y + adjustment);
    }
    if (hitBoundsMiddle != null) {
      yMovement = Math.min(yMovement, hitBoundsMiddle.y + adjustment);
    }
    if (hitBoundsRight != null) {
      yMovement = Math.min(yMovement, hitBoundsRight.y + adjustment);
    }
  } else if (yMovement < -0.0001f) {
    // Check bottom side of chef at left, middle and right
    float bottomBorder = getY() + yMovement;
    Rectangle hitBoundsLeft = getCollisionObjectBoundaries(getX(), bottomBorder);
    Rectangle hitBoundsMiddle = getCollisionObjectBoundaries( x: getX() + getWidth() / 2f,
        bottomBorder);
    Rectangle hitBoundsRight = getCollisionObjectBoundaries( x: getX() + getWidth(), bottomBorder);

    // Calculate new change in y relative to the collision object boundaries
    float adjustment = collisionSkin - getY();
    if (hitBoundsLeft != null) {
      yMovement = Math.max(yMovement, hitBoundsLeft.y + hitBoundsLeft.height + adjustment);
    }
    if (hitBoundsMiddle != null) {
      yMovement = Math.max(yMovement, hitBoundsMiddle.y + hitBoundsMiddle.height + adjustment);
    }
    if (hitBoundsRight != null) {
      yMovement = Math.max(yMovement, hitBoundsRight.y + hitBoundsRight.height + adjustment);
    }
  }
  return yMovement;
}
```

The “adjustVerticalMovementForCollision” method is the exact same copy as the “adjustHorizontalMovementForCollision” method however except for checking 3 cells east and west of the chef it checks the cells north-west, directly north, north-east, south-west, directly south and south-east of the chef. Because of this the code is almost identical except all the “getX()” methods are replaced with “getY()” and all the “getWidth()”s have become “getHeights()”s, which means the essential logic would be the same meaning any tests ran would have the same outcome as the manual tests written for adjustHorizontalMovementForCollision method.