Testing and Integration

1. Syntax Error

The following code in the LevelLoader class produces a syntax error:

```
for(int h == 0; h < fruitIndicator.size(); h++) {

	fruitIndicator.get(h).setCenterX(50);
	fruitIndicator.get(h).setCenterY(90*(h+1) + 45);
	gamePane.add(fruitIndicator.get(h), 1);
}
```

The two equal signs at `int h == 0` perform a boolean comparison, rather than setting h to 0. Using only one equal sign corrects this issue:

```
for(int h = 0; h < fruitIndicator.size(); h++) {

	fruitIndicator.get(h).setCenterX(50);
	fruitIndicator.get(h).setCenterY(90*(h+1) + 45);
	gamePane.add(fruitIndicator.get(h), 1);
}
```

2. Runtime Error

The following code in the BadIceCream4 class produces a runtime error, and more specifically, NullPointerException:

```
public void setUpdateListener() {

	updateListener = new ActionListener() {

		@Override
		public void actionPerformed(ActionEvent e) {

			keyAction.useKeys();

			for(int i = 0; i < Constants.Y_TILEROWS; i++) {
				for(int j = 0; j < Constants.X_TILECOLS; j++) {
					lvlLoader.getTileList().get(i).get(j).refresh();
				}
```

```
                        }

                        for(int j = 0; j < Constants.numFruit; j++) {
                              lvlLoader.getFruitArray().get(Constants.fruitLvl).get(j).refresh();
                        }


                        for(int k = 0; k < lvlLoader.getMonsterList().size(); k++) {
                              lvlLoader.getMonsterList().get(k).refresh();
                        }

                        lvlLoader.getTimer().refresh();
                  }
            };
      }
```

This method is called by a timer to update the game every 1ms. However, the ArrayList called by getTileList() is empty when the game starts. Forcing the program to cycle through i*j elements in the Tile ArrayList obviously will not work. The ArrayLists are only filled when a level is loaded. Having a boolean flag that indicates when a level is loaded solves this problem:

```
public void setUpdateListener() {

            updateListener = new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent e) {

                        if(Constants.levelLoaded) {

                              keyAction.useKeys();

                              for(int i = 0; i < Constants.Y_TILEROWS; i++) {
                                    for(int j = 0; j < Constants.X_TILECOLS; j++) {
                                          lvlLoader.getTileList().get(i).get(j).refresh();
                                    }
                              }

                              for(int j = 0; j < Constants.numFruit; j++) {
                                    lvlLoader.getFruitArray().get(Constants.fruitLvl).get(j).refresh();
```

```
                        }


                        for(int k = 0; k < lvlLoader.getMonsterList().size(); k++) {
                              lvlLoader.getMonsterList().get(k).refresh();
                        }

                        lvlLoader.getTimer().refresh();
                  }
            }
      };
}
```

3. Logic Error

To center the level, two constants X_OFFSET and Y_OFFSET are used to make sure that the left and right edges of the level are equidistant from the left and right edges of the window, and that the same applies to the top and bottom edges of the level. Below is code from the Tile class which handles the constants:

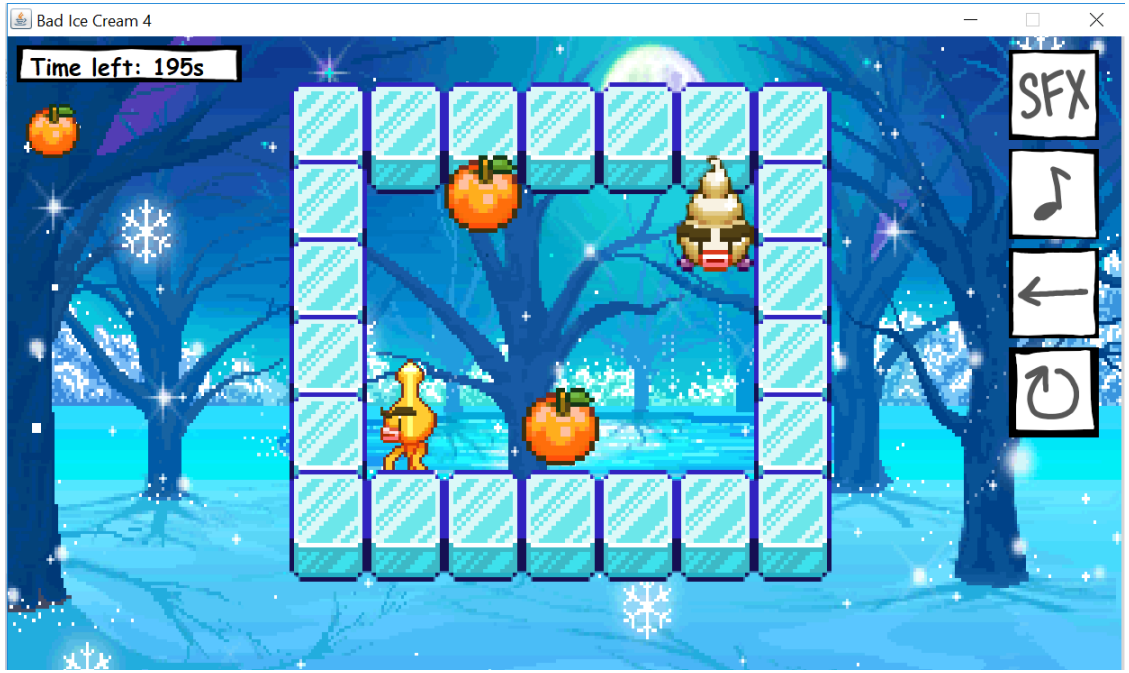
```
public void setCenterX(double xCoord) {

      centerX += xCoord + (int)Constants.X_OFFSET;
      setX((int) (centerX - getWidth()/2.0));
}

public void setCenterY(double yCoord) {

      centerY += yCoord + (int)Constants.Y_OFFSET;
      setY((int) (centerY - getHeight()));
}
```

Loading the level, the tiles are out of place and enemies move off-screen:

SetCenterX is repeatedly called every 1ms to update the locations of enemies, fruits, and characters when they move. Because it continually adds the constant to the coordinate instead of just once, the x and y coordinates of moving objects will act haphazardly. To remedy this, the constants are used only during the initialization of each tile, and during the goal setting of each tile, where the program determines where the tile should move to. Code starts on the next page.

```
	public void setCoordinates(boolean isFruit) {

		if(isFruit) {
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)(tileY*Constants.tileSize + Constants.tileSize/2.0 + Constants.Y_OFFSET));
		} else {
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)((tileY + 1)*Constants.tileSize + Constants.Y_OFFSET));
		}

		setBounds((int)x, (int)y, getWidth(), getHeight());
		setRefresh(true, 0);
		goalX = getCenterX();
		goalY = getCenterY();
	}

	public void setCenterX(double xCoord) {

		centerX = xCoord;
		setX((int) (centerX - getWidth()/2.0));
	}

	public void setCenterY(double yCoord) {

		centerY = yCoord;
		setY((int) (centerY - getHeight()));
	}

	public void setGoalX(double xGoal) {
		goalX = xGoal + (int)Constants.X_OFFSET;
	}

	public void setGoalY(double yGoal) {
		goalY = yGoal + (int)Constants.Y_OFFSET;
	}
```

The fixed game looks like this:



Game Outcomes

Win:

Loss: