<u>TESTING FEATURES FOR TIGERS</u>

Unit testing was carried out to check whether or not a tiger could have been placed in a lake or on a road. If a tiger was placed, check if it is scored for testing purposes.

A **main function** was created in the **Features function** so that unit testing could be carried out. Multiple combinations of legal tile merges were tested to make sure that there weren't bugs that would cause us to forfeit. This played an important role in creating the AI for the tournament because it helps with the scoring of the game.

Tests can be ran by downloading the Feature.cpp, Feature.h, and TestFeature.cpp files.

1. Place all of the files in the same folder.
2. Access the folder from terminal or preferable compiler.
3. Compile the file (using Terminal):
   a. type `g++ -o test1.out TestFeature.cpp -std=c++14`
   b. type `./test1.out`

```
int main(){

// Starting at row 2, column 1 from left to right on "Catalog of tile types".
// Creating unique Lake Features for each tile consisting of a lake.

    Feature * f1  = new Lake(false, 0);
    Feature * f2  = new Lake(false, 1);
    Feature * f3  = new Lake(false, 2);
    Feature * f4  = new Lake(false, 2);
    Feature * f5a = new Lake(true, 1);
    Feature * f5b = new Lake(true, 1);
    Feature * f6  = new Lake(true, 3);
    Feature * f7a = new Lake(true, 1);
    Feature * f7b = new Lake(true, 1);
    Feature * f8  = new Lake(true, 3);
    Feature * f9  = new Lake(true, 3);
    Feature * f10 = new Lake(true, 3);
    Feature * f11 = new Lake(true, 3);
    Feature * f12 = new Lake(true, 3);
    Feature * f13 = new Lake(true, 3);
    Feature * f14 = new Lake(false, 1);
    Feature * f15 = new Lake(true, 3);
    Feature * f16 = new Lake(true, 3);
    Feature * f17 = new Lake(false, 2);
    Feature * f18 = new Lake(false, 2);
    Feature * f19 = new Lake(true, 3);
    Feature * f20 = new Lake(true, 3);
    Feature * f21 = new Lake(false, 1);

// Merging multiple combinations of 'lake' tiles to check whether or not a tiger can be placed.
// If a tiger can be placed, check if it is scored for testing purposes.

//TEST CASE: 1
    /*f1->merge(f2);
    f1->addTiger(1);
    int a=0;
    int b=0;
    f1->canPlaceTiger();
```

```
	f2->canPlaceTiger();
	f1->checkIfScored(a,b);
	a=0;
	b=0;
	f2->checkIfScored(a,b);
```

//**TEST CASE: 2**

```
	f3->merge(f4);
	f3->addTiger(1);
	f3->canPlaceTiger();
	f4->canPlaceTiger();
	f3->checkIfScored(a,b);
	f4->checkIfScored(a,b);
```

//**TEST CASE: 3**

```
	f5->merge(f6);
	f5->addTiger(1);
	f5->canPlaceTiger();
	f6->canPlaceTiger();
	f5->checkIfScored(a,b);
	f6->checkIfScored(a,b);*/
```

//**TEST CASE: 4**

```
	/*f7a->merge(f8);
	f7a->addTiger(1);
	int a = 0;
	int b = 0;
	f7a->canPlaceTiger();
	f8->canPlaceTiger();
	f7a->checkIfScored(a,b);
	a=0;
	b=0;
	f8->checkIfScored(a,b);
```

//**TEST CASE: 5**

```
	f7b->merge(f9);
	f7b->addTiger(1);
	f7b->canPlaceTiger();
	f9->canPlaceTiger();
	f7b->checkIfScored(a,b);
	f9->checkIfScored(a,b);*/
```

//**TEST CASE: 6**

```
	/*f10->merge(f11);
	f10->addTiger(1);
	int a = 0;
	int b = 0;
	f10->canPlaceTiger();
	f11->canPlaceTiger();
	f10->checkIfScored(a,b);
	a=0;
	b=0;
	f11->checkIfScored(a,b);*/
```

//**TEST CASE: 7**

```
	/*f12->merge(f13);
```

```
        f12->addTiger(1);
        int a = 0;
        int b = 0;
        f12->canPlaceTiger();
        f13->canPlaceTiger();
        f12->checkIfScored(a,b);
        a=0;
        b=0;
        f13->checkIfScored(a,b);*/

//TEST CASE: 8
        /*f14->merge(f15);
        f14->addTiger(1);
        int a = 0;
        int b = 0;
        f14->canPlaceTiger();
        f15->canPlaceTiger();
        f14->checkIfScored(a,b);
        a=0;
        b=0;
        f15->checkIfScored(a,b);*/

//TEST CASE: 9
        /*f16->merge(f17);
        f16->addTiger(1);
        int a = 0;
        int b = 0;
        f16->canPlaceTiger();
        f17->canPlaceTiger();
        f16->checkIfScored(a,b);
        a=0;
        b=0;
        f17->checkIfScored(a,b);*/

//TEST CASE: 10
        /*f18->merge(f19);
        f18->addTiger(1);
        int a = 0;
        int b = 0;
        f18->canPlaceTiger();
        f19->canPlaceTiger();
        f18->checkIfScored(a,b);
        a=0;
        b=0;
        f19->checkIfScored(a,b);*/

//TEST CASE: 11
        /*f20->merge(f21);
        f20->addTiger(1);
        int a = 0;
        int b = 0;
        f20->canPlaceTiger();
        f21->canPlaceTiger();
        f20->checkIfScored(a,b);
        a=0;
        b=0;
        f21->checkIfScored(a,b);*/

//TEST CASE: 12
        /*f5a->merge(f6);
        f5a->addTiger(1);
        int a=0;
        int b=0;
        f5a->canPlaceTiger();
        f6->canPlaceTiger();
        f5a->checkIfScored(a,b);
        a=0;
        b=0;
        f6->checkIfScored(a,b);*/

//TEST CASE: 13
        /*f6->merge(f3);
        f3->merge(f8);
        f6->addTiger(1);
        f3->canPlaceTiger();
        f3->addTiger(1);
        int a=0;
        int b=0;
        f6->canPlaceTiger();
        f3->canPlaceTiger();
        f8->canPlaceTiger();
        f6->checkIfScored(a,b);
        a=0;
        b=0;
        f6->checkIfScored(a,b);

//TEST CASE: 14
        f7b->merge(f8);
        f7b->addTiger(1);
        f7b->canPlaceTiger();
        f8->canPlaceTiger();
        f7b->checkIfScored(a,b);
        f8->checkIfScored(a,b);*/


// Road Tile Features
// Creating unique Road Features for each tile consisting of a road.

        Feature * g1   = new GameTrail(true);
        Feature * g2a  = new GameTrail(true);
        Feature * g2b  = new GameTrail(true);
        Feature * g2c  = new GameTrail(true);
        Feature * g2d  = new GameTrail(true);
        Feature * g3   = new GameTrail(false);
        Feature * g4   = new GameTrail(false);
        Feature * g5a  = new GameTrail(true);
        Feature * g5b  = new GameTrail(true);
        Feature * g5c  = new GameTrail(true);
        Feature * g6   = new GameTrail(false);
        Feature * g7   = new GameTrail(false);
        Feature * g8   = new GameTrail(false);
        Feature * g9   = new GameTrail(false);

        Feature * g10  = new GameTrail(false);
        Feature * g10  = new GameTrail(false);
        Feature * g11  = new GameTrail(false);
        Feature * g12  = new GameTrail(true);
        Feature * g13a = new GameTrail(true);
        Feature * g13b = new GameTrail(true);
        Feature * g13c = new GameTrail(true);
        Feature * g14a = new GameTrail(true);
        Feature * g14b = new GameTrail(true);
        Feature * g14c = new GameTrail(true);
        Feature * g15  = new GameTrail(false);
        Feature * g16  = new GameTrail(false);
        Feature * g17  = new GameTrail(true);
        Feature * g18  = new GameTrail(true);
        Feature * g19  = new GameTrail(true);
```

// Testing the completeness of a road and whether a tiger can be placed and checking the score.

//TEST CASE: 1

```
        /*g1->merge(g2a);
        g1->addTiger(1);
        g2a->addTiger(1);
        int a=0;
        int b=0;
        g1->canPlaceTiger();
        g2a->canPlaceTiger();
        g1->checkIfScored(a,b);
        a=0;
        b=0;
        g2a->checkIfScored(a,b);*/
```

//TEST CASE: 2

```
        /*g6a->merge(g7a);
        g6a->addTiger(1);
        int a = 0;
        int b = 0;
        g6a->canPlaceTiger();
        g7a->canPlaceTiger();
        g6a->checkIfScored(a,b);
        a=0;
        b=0;
        g7a->checkIfScored(a,b);*/
```

//TEST CASE: 3

```
        /*g1->merge(g12);
        g1->addTiger(1);
        int a = 0;
        int b = 0;
        g1->canPlaceTiger();
        g12->canPlaceTiger();
        g1->checkIfScored(a,b);
        a=0;
        b=0;
        g12->checkIfScored(a,b);*/
```

//TEST CASE: 4

```
        /*g1->merge(g2a);
```

```
	g2b->merge(g6);
	g6->merge(g7);
	g7->merge(g8);
	g8->merge(g2c);
	g1->addTiger(1);
	g6->addTiger(1);
	g8->addTiger(1);
	int a = 0;
	int b = 0;
	g1->canPlaceTiger();
	g2a->canPlaceTiger();
	g2b->canPlaceTiger();
	g2c->canPlaceTiger();
	g6->canPlaceTiger();
	g7->canPlaceTiger();
	g8->canPlaceTiger();
	g1->checkIfScored(a,b);
	g2a->checkIfScored(a,b);
	g6->checkIfScored(a,b);
	g8->checkIfScored(a,b);
	a=0;
	b=0;
	g7->checkIfScored(a,b);
	g2b->checkIfScored(a,b);
	g2c->checkIfScored(a,b);*/
```

//TEST CASE: 5

```
	/*g18->merge(g11);
	g11->merge(g19);
	//g18->addTiger(1);
	g11->addTiger(1);
	g19->addTiger(1);
	int a=0;
	int b=0;
	g18->canPlaceTiger();
	g11->canPlaceTiger();
	g19->canPlaceTiger();
	g18->checkIfScored(a,b);
	g11->checkIfScored(a,b);
	g19->checkIfScored(a,b);*/
```

//TEST CASE: 6

```
	/*g18->merge(g11);
	g11->merge(g19);
	//g18->addTiger(2);
	g11->addTiger(2);
	g19->addTiger(2);
	int a=0;
	int b=0;
	g18->canPlaceTiger();
	g11->canPlaceTiger();
	g19->canPlaceTiger();
	g18->checkIfScored(a,b);
	g11->checkIfScored(a,b);
	g19->checkIfScored(a,b);*/
```

//TEST CASE: 7

//TEST CASE: 7

```
	g1->merge(g2a);
	g2b->merge(g6);
	g6->merge(g7);
	g7->merge(g8);
	g8->merge(g2c);
	g1->addTiger(2);
	g6->addTiger(2);
	g8->addTiger(2);
	int a = 0;
	int b = 0;
	g1->canPlaceTiger();
	g2a->canPlaceTiger();
	g2b->canPlaceTiger();
	g2c->canPlaceTiger();
	g6->canPlaceTiger();
	g7->canPlaceTiger();
	g8->canPlaceTiger();
	g1->checkIfScored(a,b);
	g2a->checkIfScored(a,b);
	g6->checkIfScored(a,b);
	g8->checkIfScored(a,b);
	a=0;
	b=0;
	g7->checkIfScored(a,b);
	g2b->checkIfScored(a,b);
	g2c->checkIfScored(a,b);
```