CSCD Post Task 1 Report

Chris Park

1) Changes to EntityDog:

Class name (Line 4)
*Constructor (Line 8***)**
*****Old Code*****

```
public class EntityDog extends Entity
{
	private String attackType;

	public EntityDog(String n)
	{
		super(n);
		attackType = "biting";
	}
```

*****New Code*****

```
public class EntityDawg extends Entity
{
	private String attackType;

	public EntityDawg(String n)
	{
		super(n);
		attackType = "biting";
	}
```

*****Old Code*****

```
public void initiateAttack(Entity ent)
	{
		if(ent.getClass() == EntityGhost.class)
		{
			System.out.print(name +
				" can't attack that.");
			return;
		}
```

*****New Code*****

```
public void initiateAttack(Entity ent)
	{
		if(ent.getClass() == EntityGhost.class ||
```

```
                ent.getClass() == EntityAlien.class)
        {
                System.out.println(name +
                        " can't attack that.");
                return;
        }
```

2) Creation of EntityCat:

Copied EntityDog
Replaced Class name, constructor, attackType, and move type (Line 4, 8, 11, 17)
Added check for EntityAlien (Line 23)

*****Old Code*****

```
public class EntityDog extends Entity
{
        private String attackType;

        public EntityDog(String n)
        {
                super(n);
                attackType = "biting";
        }

        public void move()
        {
                System.out.println(name +
                        " moves by walking.");
        }

        public void initiateAttack(Entity ent)
        {
                if(ent.getClass() == EntityGhost.class)
                {
                        System.out.print(name +
                                " can't attack that.");
                        return;
                }
```

*****New Code*****

```
public class EntityCat extends Entity
{
        private String attackType;

        public EntityCat(String n)
        {
                super(n);
                attackType = "claw";
```

```
	}

	public void move()
	{
		System.out.println(name +
			" moves by slinking.");
	}

	public void initiateAttack(Entity ent)
	{
		if(ent.getClass() == EntityGhost.class ||
			ent.getClass() == EntityAlien.class)
		{
			System.out.println(name +
				" can't attack that.");
			return;
		}
```

3) Creation of EntityAlien:

Copied EntityDog
Replaced class name, constructor, attackType, move type (Line 4, 8, 11, 17)
Removed checks for can't attack (Line 22-27)

*****Old Code*****

```
//Chris Park
import java.util.*;

public class EntityDog extends Entity
{
	private String attackType;

	public EntityDog(String n)
	{
		super(n);
		attackType = "biting";
	}

	public void move()
	{
		System.out.println(name +
			" moves by walking.");
	}

	public void initiateAttack(Entity ent)
	{
		if(ent.getClass() == EntityGhost.class)
		{
```

```
                System.out.print(name +
                        " can't attack that.");
                return;
            }

            System.out.println(name +
            "initiated " + attackType +
            " against " + ent.getID() + ".");

            ent.receiveAttack(this);
        }
}
```

*****New Code*****

```
//Chris Park
import java.util.*;

public class EntityAlien extends Entity
{
        private String attackType;

        public EntityAlien(String n)
        {
            super(n);
            attackType = "raygun";
        }

        public void move()
        {
            System.out.println(name +
                    " moves by teleporting.");
        }

        public void initiateAttack(Entity ent)
        {
            System.out.println(name +
            " initiated " + attackType +
            " attack against " + ent.getID() + ".");

            ent.receiveAttack(this);
        }
}
```

4) How would expanding Entities further from EntityCat effect scalability and effort involved?

There would be a moderate amount of code duplication, and depending on attack constraints some added checks or even removed ones. New class files would need to be saved and the old ones removed. Other than the code duplication, the checks could become a huge problem as the number of child

classes expanded.

5) How confident are you in your solution? Does it conform to the constraints?

I feel that although my design has been shown to be flawed. It still effectively adheres to the constraints set forth in the instructions.

6) Could someone else understand your code well enough to modify it?

Yes, I believe that in its current state, with the low number of child classes currently in implementation, and with no additional constraints outside of the types in the pdf, it can be understood easily and modified.

7) How many checks are handled at compile time?

100% of the checks in the initiateAttack method of all classes are run at execution time.

8) Do you think your solution would be considered professional?

Given the lack of compile time checking, and a design that lends itself to code duplication and possible non linear expansion of checks; Not as professional as I'd hoped it might be.

9) What would you change about the design to improve it?

Though I'm not sure on the specifics of some of the implementation, there definitely should be a better way to check constraints in initiateAttack that prevents successive child classes from spiraling into the spaghetti code apocalypse. A lot of code duplication could also be avoided by moving it to the abstract class or possibly moving it to an interface. There are also several places where public methods could be changed to better limit availability outside the classes.

10) Who did you take CS210 and 211 with?

I took both of these classes at Spokane Falls Community College. My instructor was John Mill.