Advance Graphic Report
Assignment 3
Ibrahim Mushtaq

Report:

This assignment explores creating a collision avoidance system, by simulating crowd simulation using monkeys. The goal was simple, randomly generate two tribes of monkey opposite of each other, and let them try to get to their goal. This would involve colliding with obstacles, and other monkeys.

The plan that I had in mind, is the following:

1. Get all the model rendered on the screen
2. Once all the model is rendered on the screen, generate Bounding Box (AABB) for each model, and obstacle
3. Figure out, if the two box collides or not.
4. To test, #3, generate monkey such that they don't generate on top of each other, and the obstacles
5. Add movements to the monkey, such that it goes towards the goal
6. Check if there are collision, and move accordingly
7. Make it boid like object such that it will match the speed with other monkeys within the tribes

This plan seemed easy, since its like a checklist. The first plan is easy, read the file, bind it to a VAO and when its time to draw that object, you can do so by calling the VAO.

The second plan was quite difficult. Getting a bounding box for the obstacle was hard as it is part of the original model. However, knowing that I had not a lot of time to make a good AABB generator, I decided to find out the step size of the grid, and just iterate through the file. If the y value is greater than 0, then it's an obstacle.

```
for (i = 0; i < nv;) {
       x = shapes[0].mesh.positions[i++];
       y = shapes[0].mesh.positions[i++];
       z = shapes[0].mesh.positions[i++];
       glm::vec3 vert = glm::vec3(x, y, z);

       if (y > 0){
              BoundBox b;
              b.min = glm::vec3(x - STEPS_SIZE, 0.0f, z - STEPS_SIZE);
              b.orig = vert;
              b.max = glm::vec3(x + STEPS_SIZE, y, z + STEPS_SIZE);
              bb.push_back(b);
       }

}
```

The code above describes how a bounding box is generated, I find the min vertex and max vertex. However this is for the ground. Creating a AABB for other model is quite simple.

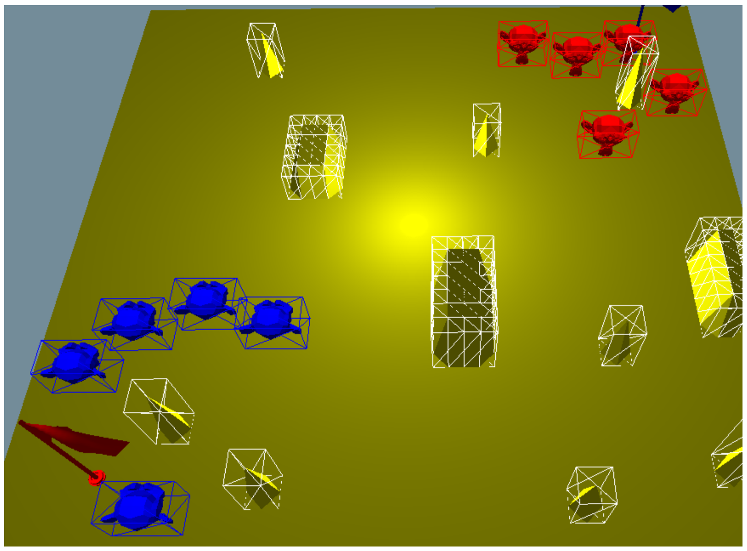
Figure 1 AABB on each object, except for the flags

Figure 1 shows the AABB outline on each object except for the flags, as that is there to show the goal of the monkey.

The third plan was really simple, to find out if the two box overlap or collide, we just compare the object 2 max is created then min, and object 1 max is greater than object 2 min. The code that handle this is shown below:

```
bool checkCollisionArray(BoundBox b1, std::vector<BoundBox> b2) {

	for (int i = 0; i < b2.size(); i++) {
		if ((b1.min.x <= b2[i].max.x && b1.max.x >= b2[i].min.x) &&
			(b1.min.y <= b2[i].max.y && b1.max.y >= b2[i].min.y) &&
			(b1.min.z <= b2[i].max.z && b1.max.z >= b2[i].min.z)) {
			return true;
		}
	}

	return false;
}
```

The easy part is done, now we need to generate the monkey such that they don't overlap with each other and the obstacle. Figure 1 shows an example of this, as they don't overlap with each other.

Adding movement is also easy, as you have the direction vector. Part 6 is the tricky one which I haven't done well. However, what I do is in the Fixed Update() function, I update the monkey position accordingly. I have a while loop that checks if there are any collision. If there are collision, then add a force to the right or left. As you will see, the monkey looks like they are teleporting. However, since there is a loop, the final position is what makes it teleport. (Hint, when running the program, there is a checkbox to see this effect's). There is a lot of improvement in this section of the code, but due to time constraint, this couldn't be achieved. However, this could does technically work, as it will avoid the obstacle, its own tribe and the other tribes. The last bit is easy, since we are trying to match the speed of the other tribe member.

The whole code could have been better. There are multiple places where I could have put a block of code into a function, and just call it. The whole, when there is a collision and move left or right could have been done better. Maybe like seeing which side is the shorter and try to move toward that side. Having the model rotate such that its looking at the goal, instead of no rotation, and more could have been overall better.

How to use:

Just unzip the file, and run the solution. If there may be and IMGUI error, just go to the solution explore -> Resource Files -> imgui -> imgui.cpp and run the program from there.

The program still uses the FPS camera, so you can see what is really happening, and the GUI do have instructions .