Justice Frimpong (jfrimpon)

Dominic Liu (xingchel)

Computer Game Programming

September 30th, 2015

**Project 1**

**Documentation**

We implemented the following features :

Required : Wander, ReachGoal, FlockingWander, Obstacle avoidance

Extra Credit (Optional) : VShapeFormationWander, Flee

Additional Features : Animation blending, Player control of goal

**How to run the scenes:**

* Wander, FlockingWander, VShapeFormationWander: Press play
* ReachGoal and Flee: Press play and control the red cube (the goal or enemy respectively) with WASD

**Approach**

We created a script called NPCBehaviour.cs which has the bulk of our code. In both ReachGoal and Wander, we were essentially moving our characters according to accelerations calculated towards some point. Therefore, the main differences between ReachGoal, Wander, and the other behaviors is how this acceleration target is chosen.

**NPCBehavior**

The characters accelerate towards a position (target) with a set maximum acceleration magnitude and maximum speed. We use raycasts and overlapping spheres on the characters to determine whether he can go forward, or if an obstacle is near him. We determine a character’s acceleration by accumulating accelerations acting on him (from obstacles and his current acceleration), obstacle repulsion accelerations are weighted according to not only the obstacle’s distance, but also the obstacle’s identity; avoiding a wall should have a higher priority than avoiding another character. Once the character gets close to the target, he starts to slow down until he stops on the target.

**ReachGoal**

We get the position of a public gameObject (the red cube) that we assign as the character’s goal and control using the cursor keys, and set the target to that position. Then, NPCBehaviour calculates and sets the acceleration towards that target. In the simulation, watch as the samurai race to the red cube, repelling each other until one finally stops on the cube, while the others flock around him helplessly.

**Wander**

We begin by choosing a targetDir (Vector3), which represents the direction we want the samurai to eventually accelerate towards. Then, we rotate tempDir (Vector3) by a certain angle every frame, and set acceleration to point towards tempDir. The result is that the samurai’s acceleration rotates until it is close to pointing in the same direction as targetDir, at which point we calculate a new targetDir and repeat the process. We choose a targetDir as follows: if there’s no obstacle interfering with his current acceleration, then we choose a random angle with no constraints; if there is an obstacle near him, we set him to accelerate and rotate faster away from the obstacle.

**FlockingWander**

There is one leader who wanders, and everyone else performs ReachGoal on the wanderer. The wandering leader is differentiated from the followers, and leaders ignore the positions of their followers. Thus, the leader can wander without being repelled by the followers, who must still avoid him.

**VShapFormationWander**

Again, one samurai is the wandering leader, while the others are followers, performing the reach goal behavior on positions diagonally to the left-behind and right-behind of the previous goal. The followers assign themselves ID’s (static int), and use it to scale their distances from the leader on the V wings, spaced by ⅔ rayDist. The leader’s turning speed has been lowered to allow the followers to keep up with their positions in the V, especially those farthest away from the front. As in FlockingWander, the leader wanders without being repelled by the followers.

**Flee**

The samurai now flee from the red cube controlled by the cursor keys. As long as the character is far enough from the red cube, he simply wanders. If he is close enough to the red cube, he gets the acceleration determined by ReachGoal and inverts it to accelerate in the opposite direction. We then re-apply the obstacle avoidance acceleration calculation to make him avoid obstacles. There is also an intermediate zone between the fleeing and wandering zones. Entering this zone will cause him to perform the behavior associated with the zone he was most recently in. This is to prevent him from constantly alternating between fleeing and wandering when on the edge of the fleeing zone.