CMPS 146: Game A.I. Final Project:

Hunter Predator Prey

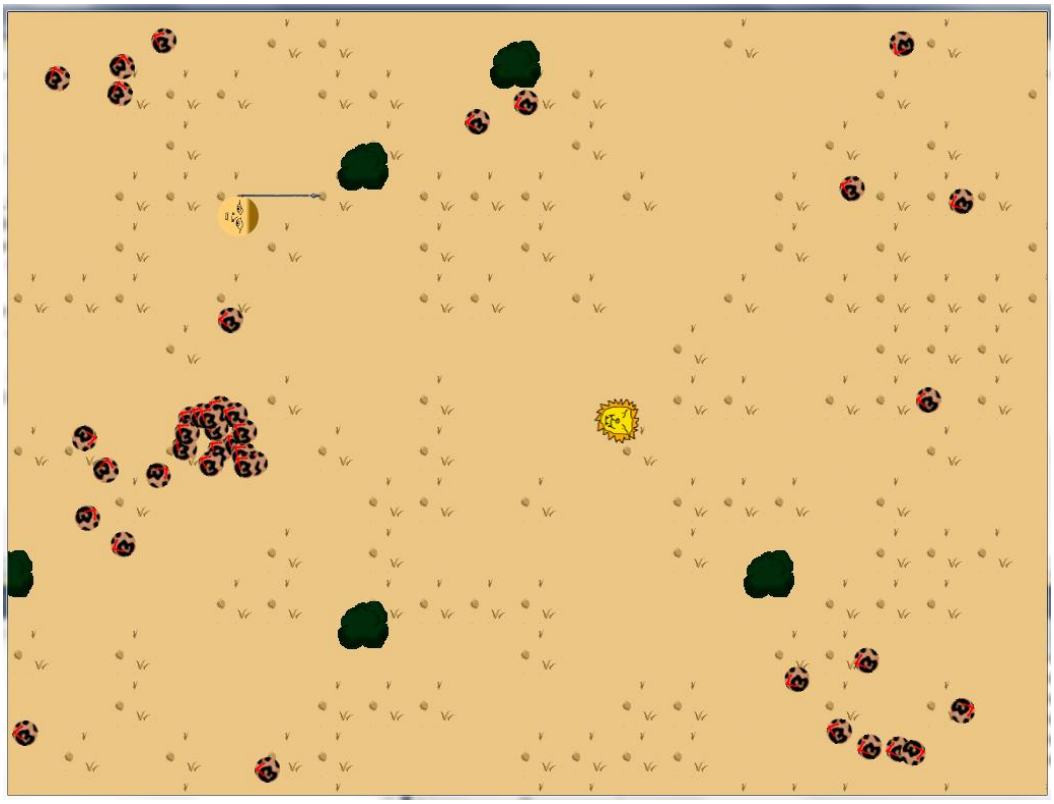
B. Stewart Bracken Ranveer Dhaliwal Jacob Rushing Matthew Collins

Introduction :

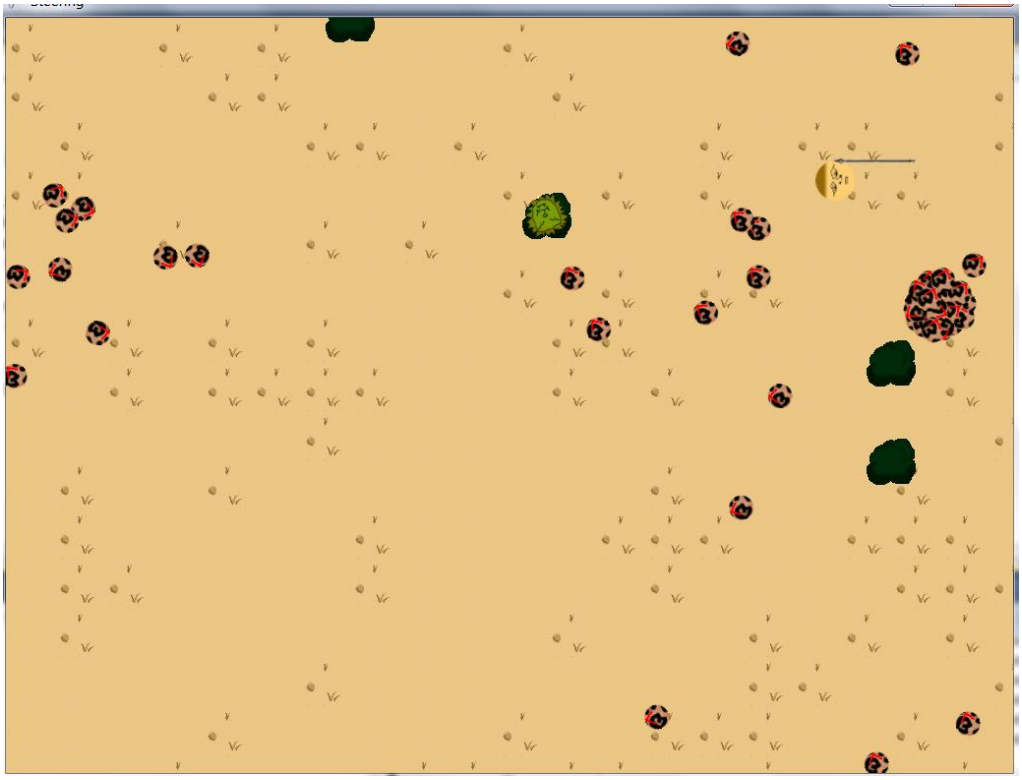
In our game, Hunter Predator Prey, we wanted to create an environment where the deer roam around, the lion preys, and the player/hunter can hunt. We set out to create a believable hunter/prey/predator relationship and the grey areas in between. Something that seems like a predator can also become the prey and vice versa. So we made this world inhabited by deer, a lion, and a hunter, all of which occupy the same area and live by the same rule: survival of the fitness. Our idea started as a simple simulation of nature but then it grew to add in the human element for more interesting gameplay. The game works simply: the deer eat grass and roam as deer do, the lion hunts either the deer or the player, and the hunter can hunt deer or the lion if they seek a more challenging adversary. To actually implement this we used a brilliant blend of Hierarchal Finite State Machines and steering behaviors with a slight use of a naïve influence like fear system. The different states return different actions which output steering vectors to each entity and that's how they move around in the world. Overall, our game worked out well, we got praise for our cool A.I. but we hold the opinion that the actual game aspect could have been better.

Player Experience:

Our game is completely made up of our A.I. so the interaction was the key to success. We needed the animals to behave well and to react to each other in an appropriate manner. As components in games interact or react with the player and with each other, they tend to feel like they actually exist rather than they just take specific actions at specific times like a robot. We have to turn that robot to seem like it's a human. In other games for example, they are enemy types that do nothing except attack the player when he comes near. Those enemies do not seem like living creatures, instead they seem like automated instruments which the player only has to get through. We wanted our A.I to seem real by having behaviors that could match up to real animal behaviors, where they would hunt when hungry, flee when scared, and hang out when they do not feel threatened. We can easily say we accomplished this goal pretty well. What the deer do depends on their fear level, which goes up when a lion is near or when threatened by the hunter, it also influences the nearby deer's fear as well. Here we can see the lion just hanging out, not hungry, and the deer are wandering as they please:



It is really hard to convey the fluid motion that they are taking and how they flock or wander around in a screenshot so please play our game to see the beautiful motion in action. As our lion gets hungrier, he starts to hunt his prey. At low levels of hunger he goes after the deer, spreading them out and picking the most viable target. If he fails, which can happen even to an expert lion, he will then hide in a bush to pounce deer for an easy snack, as we see below:



If even that fails by bad luck or being thwarted by the player, the lion will become desperate and begin to hunt the hunter. Since the hunter is weak and slow, the lion can make quick work of him, killing him in two quick bites. Finally we come to the player, the hunter has his trusty spear that he can stab or throw. The player can intimately affect the deer and lion by killing them or the player could even just sit and watch the show as the lion hunts. The player experience is definitely shaped by their own actions, but the best part is the A.I. will shape the player experience as well depending on the players own actions. We feel like we got a good balance between the A.I. playing the game themselves and the player affecting the world which the lion and deer are a part of.

Technical Implementation:

We actually started by using a grid system for the animals and players to move around but after a while we felt like that would not be cool enough nor flow well enough, so with some advice from our awesome T.A., Peter, we decided to go with steering to move around our creatures. We first began by just messing with the steering code from the book and trying to make things move. Stewart spearheaded the steering side of our game.

"Steering was a difficult task implementing. However the result was very elegant when it all came together. Steering behavior classes all extend a steering interface, which has three overloads to the getSteering() method which always returns a SteeringOutout struct which simply contains the linear and angular change calculated by the particular steering class for the entity calling getSteering(). The best part about my implementation is that SteeringOutputs can be strung together. For example, a deer during flocking will string together getSteering() calls from separation, cohesion, and velocity match using the + operator that I overloaded to add the linear values and angular values together.

Each Action returned by the FSM and HFSM Update()s returns a string together SteeringOutput which is passed down to the Entity physics engine that adds the appropriate linear and angular values and caps them by the particular Entities MaxSpeed & Acceleration. Stringing together the flocking actions all combine into one single SteeringOutput at the base Entity level which finally applies those changes to the position and orientation of the entity.

I experimented with many different steering behaviors during development, but the final release contains 13 unique steering behaviors, some of which extend other steering behaviors. Most of these steering behaviors accept a character representing the entity being changed, and a target which is where the steering will base calculations on. However, some steering behaviors, such as when implementing flocking, require a list of targets. Conveniently the getSteering() method has overloads to run with parameters:

1. Just a character (LookWhereYourGoing, Wander)

2. Character & single target (Align, Arrive, Cohesion, Face, Flee, Pursue, Seek, Separation, Velocity Match)

3. Character & List of targets (Arrive, Cohesion, Flee, Separation, Velocity Match)

This is where I departed from the book's implementation a bit. The book creates a new Steering class for each steering a particular entity will use and links it directly with that one entity and it's target. For our game, it seemed unreasonable to create 40 different Wander behaviors for each of the 40 deer. I wanted something more dynamic and general purpose and less memory heavy. So I created a public and static Steerings Blackboard that any entity or action in the game can access. This way, there exists one wander for all deer, one lookWhereYourGoing for everyone, etc.

Our group was very happy with the way the steering turned out, most of the behaviors were smooth and nice to watch, especially the deer flocking. The biggest problem I had is that I could not figure out how to make wander work well. In the release, when the lion or any deer wander they usually twitch quite a bit. When I tweaked the wander variables at all the wander behavior would usually stop working completely. I think it's a deeper problem with the random values being generated because I know I coded the wander behavior correctly."

As the steering was being put together, we also created intricate FSMs for the deer and lion, which we soon found problematic then replaced them with HFSMs. Our HFSM is composed of the usual suspects: States, Transitions, Conditions, and Actions. With our States we also had substates that complied with the H part of our HFSM. The States we created were designed to show off different behavior patterns that the animals had. The deer have wander and graze which they switch between when not threatened then they transition to scared and flee and flocking when a lion is near or the hunter attacks. Please see the Deer FSM pdf included to get a good view of what the whole thing looks like. The idea was really having them do their deer thing, then get scared and flee and group up and finally return to their deer things when the threat was gone. The deer states work really well. Each of the different states return different steerings, so each update the deer would continue moving along with the combined steering until they changed states, like flee for example, then the Action would return a steering output that gave off a huge vector from the lion/hunter so the deer would run in the opposite way to safety. The deer Transitions and Conditions checked what the deer's fear level was at, then it transitioned to the appropriate state.

The deer fear was an idea we had from the very beginning, the idea consisted of having a deer be affected by the deer around it, sort of a ripple effect. We implemented this long before we got to the influence map part of the class but we realized we had a similar idea before even hearing of influence maps. What we did was, we had each deer keep a neighbors list then the fear from those neighbors would be adding by an equation into the current deer. This had a nice

group effect which we used to make the fear from the lion change how the deer act (which state they were in).

The lion had to be a true HFSM because he would first begin by hunting deer but he would later have to flee or hunt the hunter, a plain FSM would not have been the way to go. So at one point we had to overhaul our code to include a hierarchical structure to the FSM in order to accommodate the complexity that we wanted to create. The lion begins by wandering and napping intermittently then when he becomes hungry (which is done by a simple timer) he begins stalking deer. If he comes in range of a deer, he pounces to its position , not its current position but its position when he saw the deer was in range. We did it this way because this gave the lion some room for error, this way he will not just perfectly pounce every time against the deer or the hunter. There is a bug with this however, our game works by moving entities that walk off screen to the opposite side of the screen, like a circular map. If the lion pounces past the screen boundary, he will pounce back to the position that his was looking at, then he will move on to the next action. We were aware of this bug, but we decided it would take some nasty finagling and the human players might not notice it. Then if the lion gets too hungry he transitions to the next substate in which he tries to hunt the human. Again, all actions return steering vectors pointing at the correct position which each state requires. Please see the Lion HFSM flow chart pdf to get a good view of how this all fits in together.

In terms of non-A.I. implementation we had to code in the player, with his spear, bushes for the lion to hide in, and a nice tile grass generator underneath that gave us some nice visuals. The spear had a bit of trouble when we try to get the correct orientation down to match which direction the player was facing, but using the steering code we fixed that right up. Throwing and stabbing the spear also took some work but we got it to work well enough. The grass tile generation and the bushes are placed randomly throughout the space to give a cool effect. The actual visuals of our three main entities are very basic. The player is a head, the lion is a drawn lion, and the deer are actually jaguar prints but we felt like they looked nice when they moved around.

Profiling results:

Profiling the game gave some pretty understandable results. Out of all the bytes allocated for running the game, 68.37% of them were dedicated to our Entity class. Since practically everything in the game extends Entity, it's not unexpected. The fact that the algorithms the Lion (it uses one check) and the deer (they use three) all run in N-squared time also accounts for the extra space taken up. We could have optimized this a bit more, but at the time, we had no idea what we were doing in terms of group influence and since we keep the deer count relatively low, the game speed isn't really affected.

As for speed, we had no direct way of measuring how long each individual AI piece took to update. However, to account for this setback, we changed the framerate of our game to test performance impact. By default our game runs at 60 fps. At 100 frames per second, the game did not lag at all. At this rate, the game has 10 milliseconds per frame to update all the AI. Increasing the framerate to 500 fps sped up the gameplay, but also had no performance impact. At 500fps all of the AI has 2 milliseconds up update. Continuing, at 1000fps the game has 1ms to update all AI each frame. The game showed small signs of lag, but for the most part it still ran. This mean that each frame, our AI updates and executes in less than or equal to 1 millisecond. Normally AI is a small part of a larger game. I'd assume that 1ms is a reasonable amount of time dedicated to AI. In conclusion, our AI is efficient enough to not impact gameplay.

Play Testing:

Our game was really well received. We had quite a few people play test our game to see if our A.I. matched up to what we wanted it to be. Overall we believe it almost reached what we wanted but it stilled made a fun gameplay experience to the people playing.

Once again, we wanted our game to simulate a struggle between prey/predator/hunter, an uneasy separation that could blow at any moment. From our play testing results, we seemed to have an interesting lion/hunter relationship and the deer was mostly fodder for both sides. This is not too far away from what we wanted, because what we wanted really ends up being what they said. The deer really play innocent creatures that just become food, it’s the struggle between the player and the lion in which the real actions happens.

One play tester said our A.I. was really cool and he really loved the way everything flowed together (steering). He also said the game is fun because you don’t even have to play, you can just sit a watch the lion do his thing. Almost every playtester we had said that the lion was very strong and even hard to kill. Specifically, they said when the lion is on your tail it is really hard to get away from his pounces. Really, that is what we wanted because pitting a human with a spear versus a full grown lion is not in favor of the human. Another playtester did not realize that the lion would actually attack the player and once he found out, he started concentrating on just fighting the lion and ignoring the deer. This tended to happen once they understood that they could actually fight back. At this point, the players changed it from what our vision was, to what they wanted to see: a showdown with a formidable creature. So maybe this was not exactly what we wanted but it provided a good gameplay experience for our testers. Another tester said that she really liked how the creatures had behaviors and were not just static enemies that just chased and hid in a robotic way. The steering combined with our states really did a good job in simulating flow in our game.

Conclusion:

In conclusion, we feel like we did an amiable job at what we wanted to accomplish and even though it was not exactly on point we made a game that entertained people. We may not have made a perfect environment where these three have a constant struggle but we made a nice game that players focus on the relationship between the player and the lion and the fight for dominance. Our game really came together the last few weeks as the HFSM and the steering actions were refined to resemble our initial vision and it just looked awesome seeing the deer roam around and the lion hunting its prey. One thing we would have done differently is plan out the states a lot better rather than having to come up with some extra ones when we find a bug. Although, to be fair, the game ended up better because we found bugs and added code to remedy it but we would have wanted more prethought before we started. Also, we wished we could have made this into an actual game instead of this one level, one type of interaction. It would have been cool if there were different prey and predators with different behaviors. Overall, we feel like we did a great job and we are happy with the result.

Appendix:

Work Breakdown:

B. Stewart Bracken –

Planning for idea, states, and behaviors, HFSM implementation, all the steering behaviors, Lion & deer FSM fine tuning, part of the write up, all art except terrain, and flow chart diagrams

Ranveer Dhaliwal –

Planning for idea, states, and behaviors, HFSM implementation, deer manager including fear, lots of bug fixing and fine tuning of both State machines, and compiling the write up.

Jacob Rushing –

Planning for idea, states, and behaviors, Deer FSM structure, fine tuning HFSMs, a lot of work on getting the spear to work well, and part of the write up.

Matthew Collins –

Planning for idea, states, and behaviors, fine tuning the HFSMs, part of the write up, spear orientation fixing, and a lot of work on the tile terrain generations.