Emily Moeller

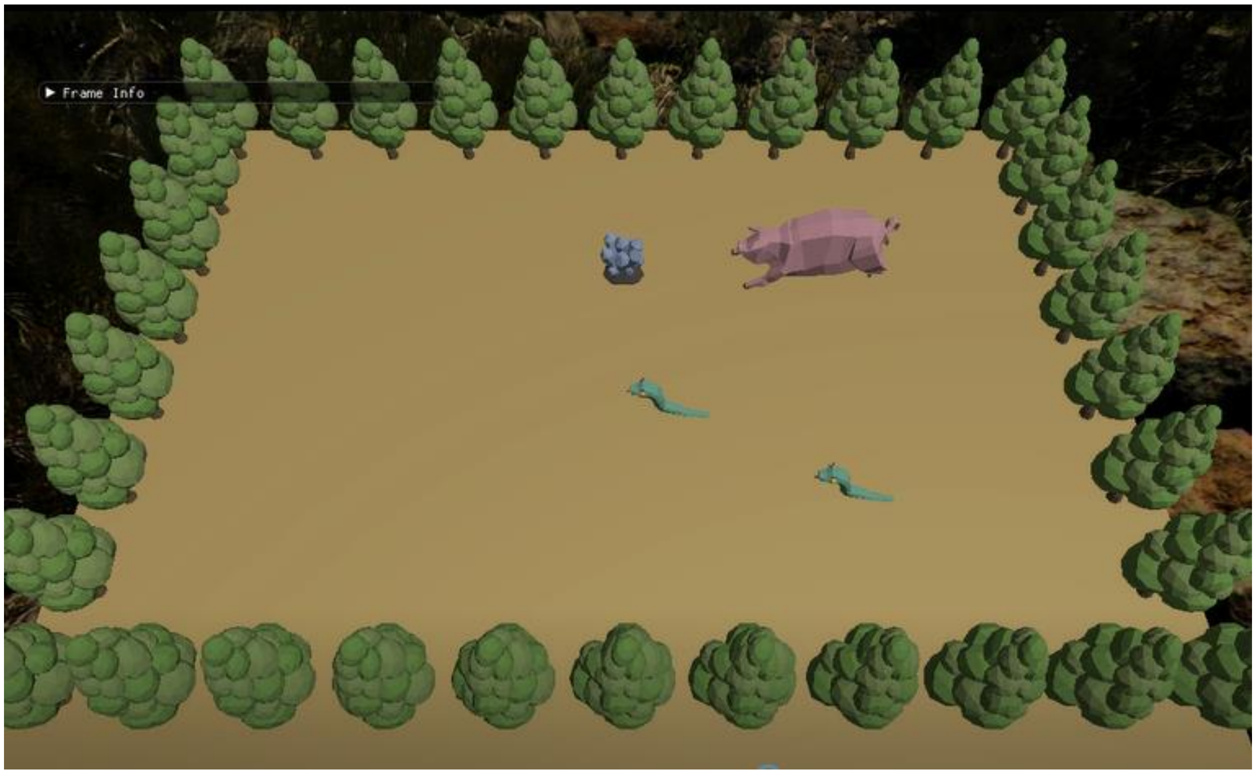
12/14/2019

Final Project Report

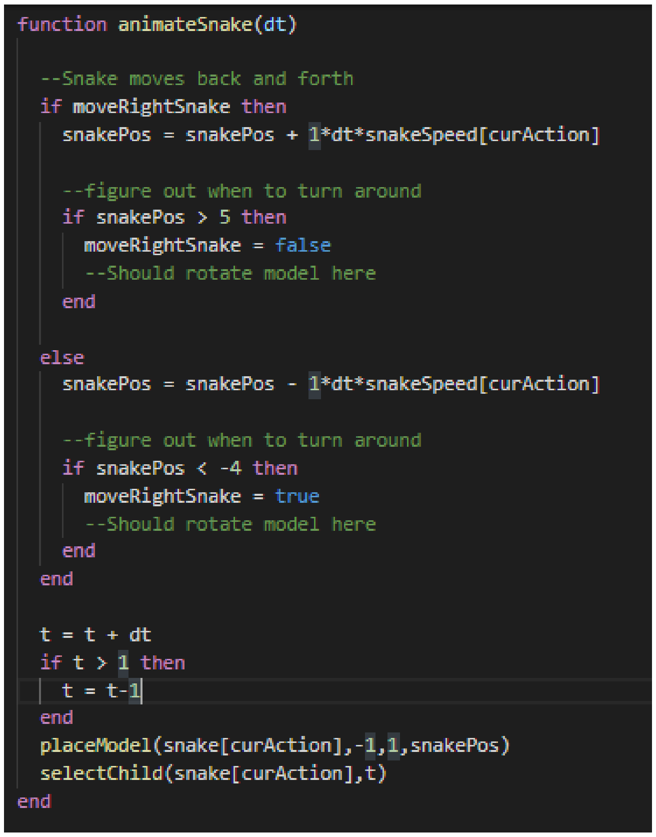
1. What features did you implement? Why did you choose those features?

After reflecting on the previous four assignments, I realized that my favorite one was working on game scripting in Assignment 3 when Nikki and I implemented the Memory game. In particular, I enjoyed the animation aspect of that project, which led me to want to work with more complicated animations. On the previous project we implemented a rotation and a shrinking animation, so in this project I wanted to work with an animation that involved rotating using multiple object files.

My original plan for this project was to create an interesting scene, with a variety of animations simultaneously occurring. This started with building off from the horse animation that was shown in class. Once I was able to get that working, I assembled folders of different animations of other animals such as a llama jumping, a snaking attacking, and a pig jumping. This led me to integrate these animated models into one object that would change to a new animated model when the tab key was pressed. Here is a screenshot after switching to the jumping pig.

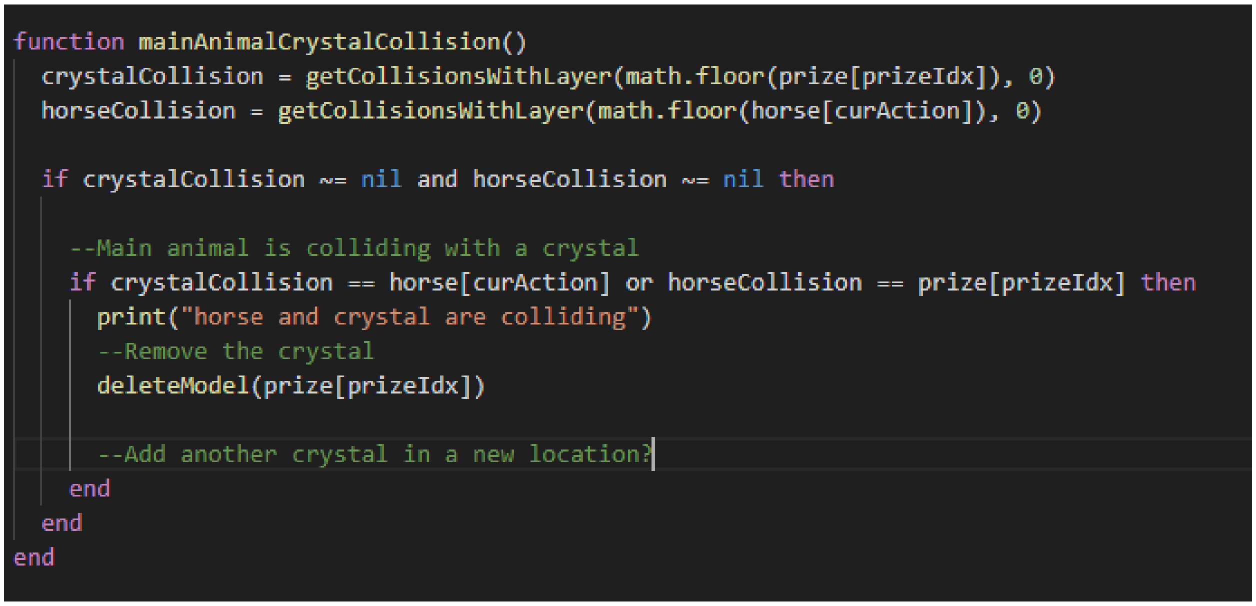


Next, I decided to make a grid outlined with trees, for the animals to run back and forth on. This involved checking the location of the animal in the animate methods and reversing the direction of the movement along the z axis, when the animal had reached the edge of the grid. The following snippet of code is from main.lua in TestHorse and it is the implementation for the snake movement.



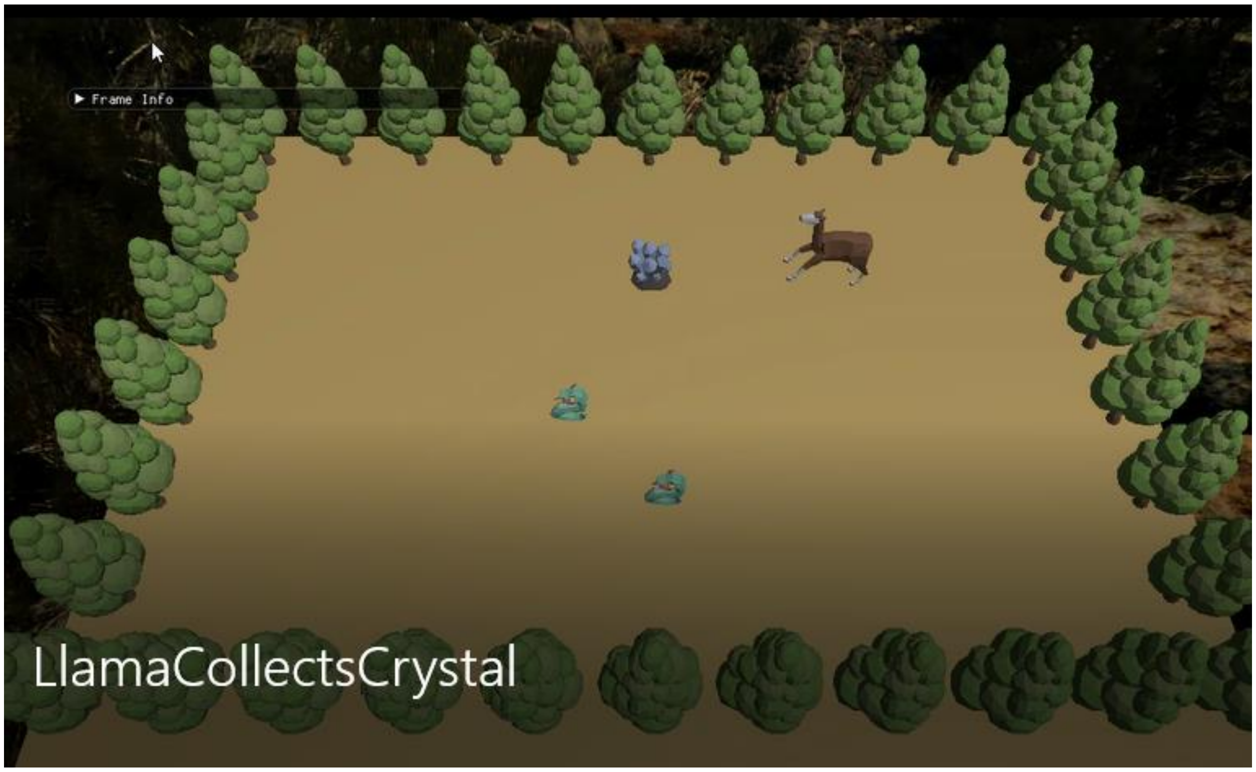
```
function animateSnake(dt)

  --Snake moves back and forth
  if moveRightSnake then
    snakePos = snakePos + 1*dt*snakeSpeed[curAction]

    --figure out when to turn around
    if snakePos > 5 then
      moveRightSnake = false
      --Should rotate model here
    end

  else
    snakePos = snakePos - 1*dt*snakeSpeed[curAction]

    --figure out when to turn around
    if snakePos < -4 then
      moveRightSnake = true
      --Should rotate model here
    end
  end

  t = t + dt
  if t > 1 then
    t = t-1
  end
  placeModel(snake[curAction],-1,1,snakePos)
  selectChild(snake[curAction],t)
end
```

It was at this point when I had the idea of creating a game where the object of the game would be to move an animal around the grid to collect crystals. The player would not only have to move to the correct location of the crystal, but they would also have to avoid running into snakes or the game would end. The concept for this game is not too dissimilar from the first game that I made, which involved collecting falling leaves by clicking on them and avoid clicking on acorns. The main differences between these two games are that this game has a main game character and that the collisions in this game are between models instead of between an object and the mouse. The first part of the game that I chose to implement was the collision handling, which was not a terribly complex task, but it ended up taking a lot of time to debug because of a misunderstanding I had about one of the collision methods, which is explained later on. The following code is the functional collision method.

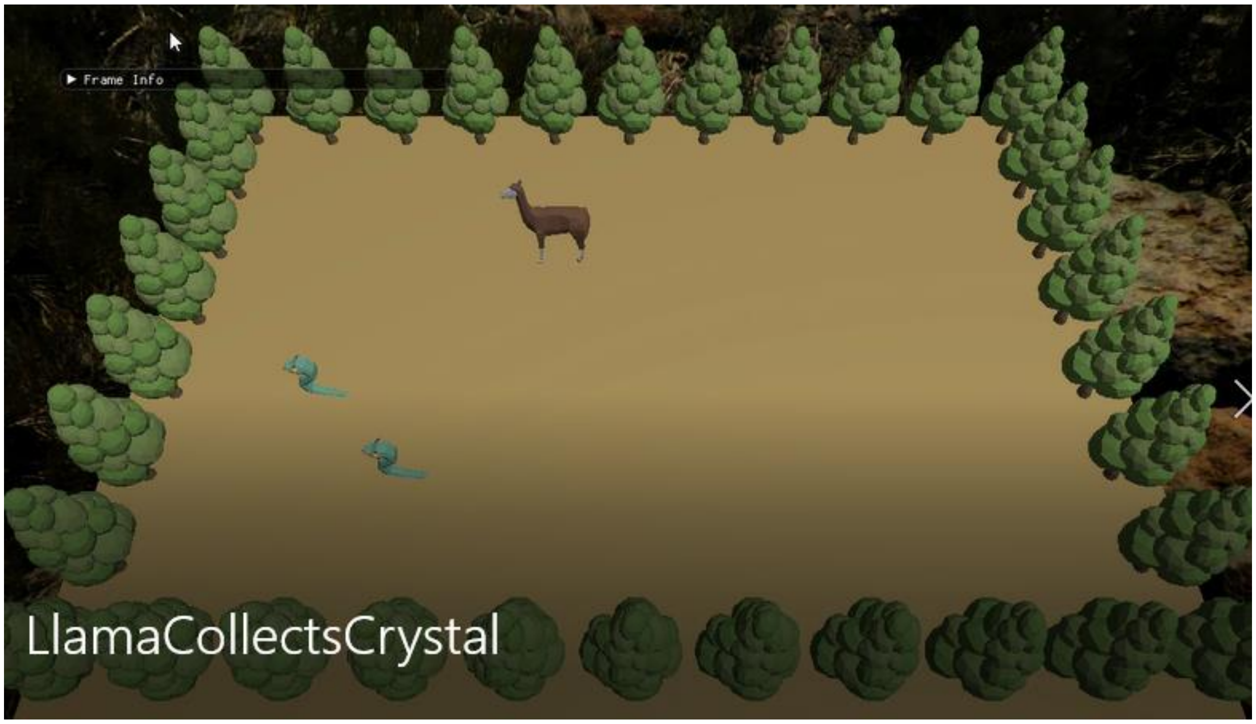


```
function mainAnimalCrystalCollision()
  crystalCollision = getCollisionsWithLayer(math.floor(prize[prizeIdx]), 0)
  horseCollision = getCollisionsWithLayer(math.floor(horse[curAction]), 0)

  if crystalCollision ~= nil and horseCollision ~= nil then

    --Main animal is colliding with a crystal
    if crystalCollision == horse[curAction] or horseCollision == prize[prizeIdx] then
      print("horse and crystal are colliding")
      --Remove the crystal
      deleteModel(prize[prizeIdx])

      --Add another crystal in a new location?
    end
  end
end
```

However, as soon as I got the collision handling to work, I found out that the functionality of changing the main animal to a different animal animation crashes the entire scene. Here is a screenshot of the functional collision handling, before the Llama collides with the crystal.



Here is the scene after the Llama has collected the crystal.



2. Algorithmically, how does your rendering implementation work. What formulas, algorithms, and techniques did you use? How scalable is your approach? What are the key bottlenecks?

In terms of scalability, my code could be structured in a better way so it is easier to create new animated and non-animated objects. This becomes especially important because if this were a real game, there would have to be a way to create many crystals and many snakes. Currently, I have a separate method for both snake animations even though their movement is practically identical, just on a different z axis. These methods could likely be combined into a more generic method so there would be less repeated code. The same principle applies to the crystals, which would be especially important if this were a fully functional game because the object of the game is to collect as many crystals as possible. It's also important that these crystals would appear in different locations so that the game would not become too predictable.

3. From a design perspective, what are some features you didn't get to? How would you improve your gameplay or demo if given more time?

If I had more time to work on this project, I would first debug the crashing that is happening when the main character changes to a new animal animation, which only happens when the collision checking is working. I would then finish implementing the other keys aspects of the game like moving the main animal with the arrow keys and tracking collisions with the snakes and the main animal to see when the game has been lost. I would also add more crystals that would appear in random locations around the grid. I would also add collision tests for the main animal and the trees, so that the main animal is unable to walk off the grid. In addition, I would make the animations look better by rotating the model when the animal moves in that direction. It would also look better if the standing version of the animal is used when the animal stops moving.

4. What were the hardest parts of the assignment from a technical perspective? From a technical implementation perspective, how would you extend your final submission if you had 2-3 more weeks to work on it?

The most challenging aspect of this assignment was handling collisions for objects. In the Memory game project, the collision handling that I implemented was for the mouse clicking objects, so I was unable to use the same methods. Instead of simply passing in an integer of the ‘layer’ to ‘getMouseClickWithLayer’ and get back the ‘hitID’ of the object in that collision ray, I now use the ‘getCollisionsWithLayer’ which takes a ‘modelID’ and a ‘layer’ as parameters and should return the ‘colliderID’. Ultimately the confusion came from not understanding what ‘getCollisionsWithLayer’ returns when there is no collision and when there is a collision. In particular, I thought it returned the ‘colliderID’ of the object that it is colliding with, but it actually is returning ‘modelID’ of that object that it is colliding with.

In terms of what I would do if I had 2-3 more weeks, I would add more interesting elements to the game to make it more challenging, such as adding crystals in random locations and implementing more unpredictable snake movement. It would be more challenging if the snake movement was dictated by the current location of the main animal, so that the player would have to avoid the snakes that would be chasing them. I think it would also be interesting to have a larger grid that potentially extends off the screen, but the camera would follow the main animal.