Sheep Saver

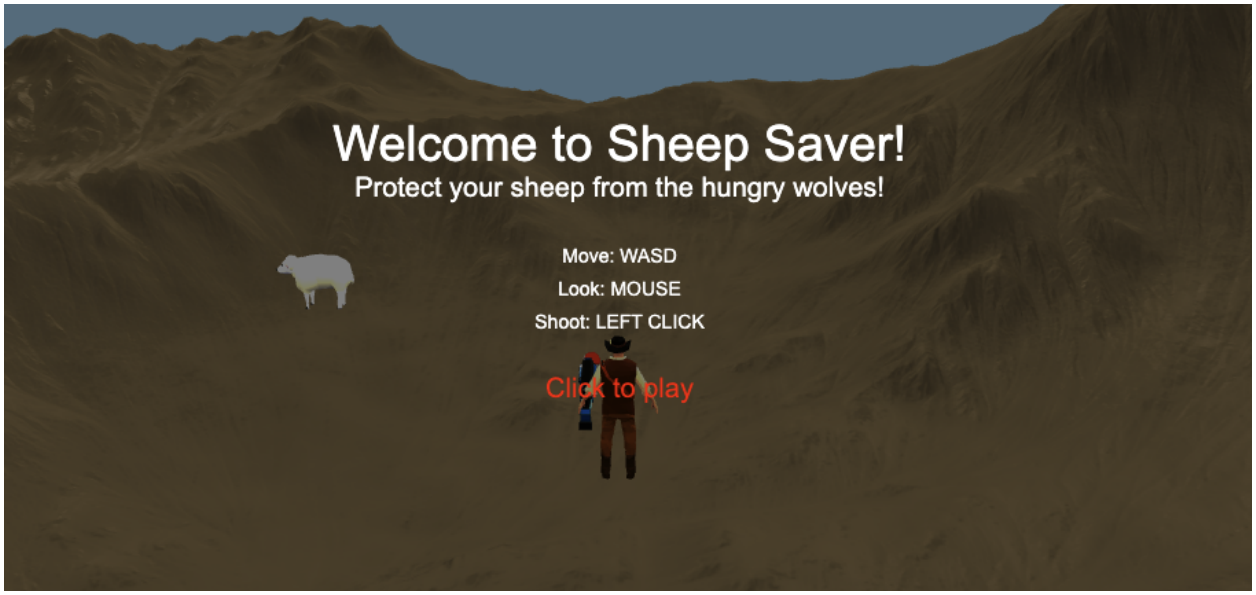
Vedant Dhopte '22, Jayson Badal '22, Justin Chang '23

Introduction

Goal

Sheep Saver is a fun to play, first person shooter game where the objective of the player is to protect their precious sheep from hungry attacking wolves for as long as possible. Located in the desert, the cowboy is accompanied by his wandering sheep and he earns points for each wolf he shoots and eliminates. The onslaught of wolves is endless. Thankfully, they have no interest in the cowboy himself, but if they reach and kill his sheep, he will lose everything that he loves.

In order to make this story come to life, we use the Three.js library and develop a 3D online mini-game complete with a desert like scene, wandering sheep and attacking wolf objects, event handlers to allow for player controls and camera movement, and a gui to interact with the game and display important information on the player HUD.



Previous Work

One previous project inspiring a lot of our gameplay is Pac-Atac. This game has very nice game mechanics that we aspire to match especially when it comes to player movement, shooting, and approaching enemies. We aim to improve upon this game in terms of design and background setting. Pac Atac takes place in a barebones gridded box environment surrounded by darkness. While fitting for the aesthetics and theme of their game, we intend to create a more complex and natural looking environment surrounding our game in order to create the feeling of playing in the desert. https://michaelf49.github.io/Pacman3D/

Another previous work we are inspired by is 3D Knockout. While not taking much inspiration from the gameplay itself, we appreciated the effort that went into creating the

surrounding environment outside the actual game platform. The surrounding ocean and mountains created a great arctic atmosphere to set the game in. We aim to create a similarly impressive desert themed terrain around our gameplay environment.
https://jlytle21.github.io/knockout/

A final source of inspiration comes from one of the Three.js examples from their website. Our movement controls as well as GUI design take much inspiration from this demo.
https://threejs.org/examples/#misc_controls_pointerlock

Approach

We set out to create a first person shooter game inspired largely by the works mentioned in the previous section combined with our own personal creative twists and designs. We implemented the game using the Three.js library in order to create our scene, add objects for the different entities in our game, create GUIs and HUD information, handle player movement and mechanics, handle object collisions/interactions and all other elements that are involved in making a solid playable 3D game experience.

We used the provided Three.js example code as a launching point to begin rendering the elements of our application. Our approach was to add all event handlers in app.js and call them in the render animation loop. We created separate files for each object type and called their movement functions within the render loop as well to update all movements other than the player. This strategy allows us to keep a modular program structure. The breakdown of file structure is as follows:

- App.js: Where we define keyboard event handlers and update the game using the onAnimationFrameHandler loop
- Components: Where we keep separate files for object, scene and light components. In objects we keep separate files for sheeps, wolves, the cowboy player and more
- Global: Where we keep global variables such as the list of enemy wolves on the screen and the player's sheep

As long as our team continues to communicate this approach should work well in keeping us organized and also in facilitating the growth of our project as it will be more clear where to go in and make changes in the future. When it comes to individual team roles, we assign fairly large chunks of the project to each member of the team (such as building the scene, making the gui, or implementing the wolf chasing sheep mechanism) and then meet every so often in order to merge our separate branches on github. We then assign new roles and keep building our app until the next meeting.

Methodology

To successfully execute our approach, multiple distinct pieces had to be implemented. In total, one can conceptualize our project as having three separate project areas. The first consists of the scene and accompanying graphics. The second dealt with the underlying algorithms and code structure that handles gameplay. The third dealt with the camera and its adaptability and versatility during gameplay so that it may offer the best user experience possible. Each of these project areas were inextricably linked to each other and this meant that an implementation of one area must fit well with the others. In addition, each project area was relatively complex and required a thorough understanding of its components to achieve meaningful results.

Scene Building

For building the scene, our team identified two main strategies. Firstly, we could create our own custom meshes to represent various objects in our scene. Secondly, we could download prefabricated meshes from the web. Both strategies had their advantages and disadvantages. For example, custom building our own meshes offered profound personalization and customizability of a scene. That is, with custom builds, anything we imagine we can bring to life. Our team viewed this as immensely powerful and it could offer us great autonomy by giving us the opportunity to design each piece of the scene to our liking. Unfortunately, however, building custom meshes is incredibly difficult and time consuming. In fact, with the given time constraints, it may have been impossible to build all meshes in time. With respect to using prefabricated meshes, this strategy was more appealing due to its time efficiency. That is, we could build worlds and galaxies of scenes in minutes simply by downloading them from the internet. With this strategy, the only disadvantage existed with customization and personalization. That is, our team had less autonomy in designing scenes since we were using those already built online. This meant that our team had to be very careful to remain committed to our original vision of the game and not get influenced or sidetracked by the meshes found online. Ultimately, our team found multiple mesh objects online that we used to create a stunning, graphically-immersive experience for the user. In total, we downloaded a desert scene mesh, boulder meshes, star cluster mesh, cowboy mesh, gun mesh, and projectile mesh.

Implementing Gameplay

For building the underlying code that supported gameplay, there were multiple different approaches that existed. A core pillar of game mechanics was having wolf objects chase sheep objects. While this may appear simple at first glance, this task is profoundly complex when one considers the thousands of variations that exist. For example, how fast should the cowboy's gun fire to save the sheep and shoot the wolves, does one shot from the cowboy kill the sheep, should wolves vary their speed when chasing sheep, should wolves increase their speed the closer they are to the sheep, should sheep increase their speed as they are being chased, when does a sheep theoretically recognize that a wolf is approaching and what should they do about it? With all these variations, our team decided that the ones most important are: the wolves speed when

chasing sheep, the sheep's recognition of being chased by wolves, and the cowboy's ability to stop the wolves. Firstly, our team decided that the wolves' speed is of great consequence and can significantly impact the difficulty of the game. Therefore, our wolves will have varying speeds by levels while throughout each level, their speed will remain constant. Secondly, we decided that our sheep should not have any recognition of being chased by the wolves and decided that they will remain unaware of being chased and will not try to run away. Here, our thinking was that the responsibility of saving the sheep was solely that of the cowboy's. Thirdly, we decided that the cowboy's ability to both stop and kill the wolves would remain constant throughout the game. That is, to promote game simplicity for the user, one shot from the cowboy's gun would kill the wolves.

Ultimately, the implementation of these designs were supported by 2 main objects: the wolf object and the sheep object. With respect to the sheep, they were initialized and spawned randomly in the game with an intensity based on the current level. The sheep then varied its position with an offset vector that represented the magnitude of its speed. The sheeps direction was also randomly chosen and manipulated. With respect to the wolves, they were also initialized and spawned at random locations in the game. The intensity of which was determined by the current game level. The wolves' direction was determined by finding the sheep it was closest to and calculating a vector from the wolf to the sheep. This was the wolves' respective direction vector that it used when moving. This direction vector is updated at regular time intervals. With respect to the cowboy firing the gun, this was simulated with the help of a projectile object. Everytime the user fired the cowboy's gun, a projectile was initially spawned and initialized at the front of the gun barrel. This projectile's position was then updated using a direction vector. This direction was calculated from the camera view position and a large offset position directly in front of the gun. If the projectile intersected with a wolf, the wolf was killed and the projectile was deleted. If the projectile intersected with a sheep, the sheep was killed and the projectile was deleted. If the projectile intersected the boundaries of our scene, it was deleted.

Camera View

For the camera, our team had two options to consider. With our game, there existed a first person view and third person view for playing. Both options had their advantages and disadvantages. For example, first person view allows the player to feel more comfortable and immersed in the game setting. It also allows the player to imagine themselves in the game. For third person view, the player may appreciate visualizing the character they are playing as. This will allow them to have an easier time to adapt to the controls of the game. That is, in third person, the player sees the entire character and is able to appreciate the surroundings front and back at the same time. Ultimately, we decided to implement first person view due to its ability to more effectively immerse the player in the game setting. We implemented this functionality by leveraging the camera mechanics learned in class and from previous assignments. We also implemented event handlers so that either arrow keys or "WASD" can manipulate the camera's position by adding an offset to its position. In addition, we implemented the feature of changing

the camera's "look at" direction vector based on the user's cursor position. This added a tremendous level of interactivity and immersion to our game.

Measuring Success

Overall, measuring success was especially important to our team. We prioritized both tangible and intangible metrics and informed our tasks based on how well we thought we were performing. With respect to tangible metrics, as a team, we set benchmarks for ourselves. For example, we needed a quality looking scene, fluid gameplay, and fast rendering. Technically, it was not difficult to measure our progress with these metrics since success meant it worked while failure meant it did not. With programming, it is rather easy to tell when something did not work since it usually had very obvious visual signs. The real challenge came with evaluating our intangible metrics. The most significant ones to consider were the user experience and connection to the game, the game complexity and thoughtfulness of scenes, and the aesthetics of our graphics. These intangible metrics have no clear way to evaluate them. For this reason, our team made sure to create a productive space where we could critically evaluate and refine these areas to our liking.

Discussion

Alternative Approaches

We believe that the approach we took towards completing this project was a successful approach. However, we definitely could have streamlined the process better by planning out more specific intermediate goals for the game, and starting the process earlier. Additionally, instead of splitting up the project into different parts and working on each part to completion, we could have split it up into stages of complexity, finishing the skeleton of the project first before moving on to the details, cosmetics, etc. It is not entirely clear that this would be a strictly better approach, but it is definitely something to consider when working on projects of this or greater size and complexity.

In terms of file structure, we used git to share files and work synchronously, which allowed for more efficiency than all working on the same file together, which is what is recommended in earlier COS classes such as COS 226. An alternative approach would be for each members' code to be in a different file and to import each other's files. However, our approach is not only more efficient than this since there is a central file system, but it is much more organized.

Future Work

As we have not finished the game by the Dean's Date deadline, we will continue to work on it during the week before our presentation on May 14th. During this time, we hope to accomplish the task of finishing the game to a level of satisfaction from all group members. What this will look like is a game with working controls, interactions between actions and NPC objects, correct (to an extent) physics, and an aesthetically pleasing graphical user interface. In

the finished product, the player will be able to control their cowboy character in a first person setting, fending off randomly spawning wolves that are trying to eat the player's sheep. To do this, the character will be able to shoot bullets at the wolves, and rack up points by eliminating wolves. When all the sheep are eaten or the timer runs out, the game will end, and the player's score will be shown.

However, while we will have a "finished" product, there is always room for improvement. For example, we have thought about implementing a feature that allows the player to choose third-person versus first-person controls, a scoreboard feature that tracks the high scores between games, and more realistic animations for the objects in the game, such as eating or dying animations. Other interactable objects can be added to the game as well, such as power-ups, or cacti that do damage to whoever runs into them. On the more technical side of things, the game's efficiency can also be improved by only rendering what needs to be seen on the screen, for example. Aesthetically, more details can be added to the background, or possibly the ability to switch between biomes.

What We Learned

As a group, we learned a lot as we were able to put into practice a lot of the topics that we have studied throughout this course. First and foremost, we learned how to use git well and how to streamline the development process, which is extremely important in the world of software development. We also dug into different npm packages and ThreeJS modules to help get our game up and running. This is also useful, as it is rare these days to create something 100% original; there really is no need. It is not only easier, but smarter and more efficient to be able to incorporate other people's packages and modules into your own projects.

Conclusion

Did we Effectively Reach our Goal?

Overall, we were very satisfied with what we have so far, and what we aim to get done by the end of the week when we will present to the rest of the course. We have a very promising skeleton of what we envision our final product to look like. We have a scene that fits our theme well, good movement mechanics and decent working AI of our sheep and wolf objects.

Next Steps

The next steps to take are to first deal with object collisions such as the sheep dying after touching too many wolves and wolves dying after being hit with enough bullets. We also need to find some algorithm that spawns wolves at a reasonable rate. The last steps would be to polish up the graphics of the game and add animations to our characters and also incorporate sounds, music, and HUD graphic displays such as sheep health and points accumulated.

Things to Revisit

Finally, some things that need revisiting are the AI of our enemy and sheep objects so that their movements seem more natural. For example, we still need to integrate their movements with the surface of the scene. They currently walk along a constant plane low enough to make it seem like they are constantly in contact with the surface.

Contributions

We split up the work into different sections, as stated previously in this report. At the beginning, Justin Chang worked on the controls and the camera, creating the first person setup we imagined. Jayson Badal worked on the background landscape and textures, finding usable and aesthetically pleasing meshes for the elements of the game. Vedant Dhopte found meshes for the sheep and wolves in the game and coded their movement patterns, semi-random for the sheep and path-following for the wolves. As the work progressed and people finished their tasks, we discussed what the next thing to work on was. Justin Chang also created the GUI and HUD for the game and worked on the act of firing the gun, Jayson worked on the animations of the objects in the scene, and Vedant worked on the interactions between objects (i.e., bullet hitting a wolf, wolf eating a sheep). Every day, as a group we would meet to discuss what we were working on, what we were stuck on, and what we needed to focus on more. We would also meet to merge pull requests from the different branches each member was working on.

Works Cited

- Wolf Mesh
  - https://poly.google.com/view/46bXrRt8pFF
- Sheep Mesh
  - https://poly.google.com/view/9nvBoaVZuX2
- Desert Mesh
  - https://skfb.ly/6WQpn
- Gun Mesh
  - https://skfb.ly/UVsG
- Projectile Mesh
  - https://skfb.ly/TTPW

Live Demo

https://jchang19.github.io/cos426-final-proj/