Marisa Long
Kayla Fortson
Benedicte Knudson

Features Write Up: Crossy Roads

Youtube Link: https://youtu.be/SnqayA3itBA
Please Demo Game in Github for real experience because the video recorded choppy.

Interaction
User interaction in our game involves using the right and left arrow keys to move the pelican from right to left and hitting the spacebar to fly up the screen.

Movement
We have multiple moving components in our game. Our pelican moves right, left, and up per the user's manipulation. The floats on the road move either left to right or right to left according to their respective lanes.

Models
We have 4 different models included in our game. The main character, the pelican, has its own class with methods such as update state rotate right/left, and a thruster to make it fly. The roads are another class called lanes that makes it easy to draw as many roads as we like, wherever we like. This would come in handy when building out the game to contain multiple levels. The third class is the floats class, which contains the mardi gras floats that traverse the screen. Each float is its own object. Therefore, they also have their own methods within their class such as update state. All of the floats are stored in vectors which allows the user to manipulate multiple floats. This allowed us to draw many repetitive floats efficiently, control how many per lane, add additional floats if necessary, remove floats when needed, add movement to many floats, and even translate the floats around as needed. Finally, we have the grass, which is a class that randomly generates grass blades within the screen bounds using translations.

Color
We tried to choose colors appropriate to our Mardi Gras theme. The floats are the traditional Mardi Gras colors with randomly generated colored beads hanging from the sides. The pelican, grass, and lanes are all drawn in realistic colors to represent their respective objects.

Geometric Algorithms
The main geometric algorithm that our game uses is a collision test, which we called inside_outside_test. This algorithm uses bounding boxes to check if the pelican has collided with any of the floats in the lanes. If the pelican does collide with a float, it switches a boolean variable to true and makes the pelican fall down the screen. Another geometric algorithm that we employed was rotate right/left. This specifically was applied to the pelican so that it could walk in different directions and be facing the appropriate way. We also included update state in both the pelican and the floats classes to enable constant movement in the game. The floats were set to a constant velocity which increased the position of the floats each time update state

was called, giving the floats smooth and constant movement. Update state for the pelican was sensitive to user input and reflected on the screen via manipulation from the user. The thruster, or jumps was another algorithm we built for the pelican so that it can fly up the screen. When the thruster is on, it travels upwards and spreads its wings. Another algorithm that we implemented was wrapping. If the pelican exits the screen on either side, the pelican will wrap back around the screen. This was implemented by converting the pelican's modelview into worldview, seeing if the coordinates were outside of the screen's bounds, and then translating the pelican accordingly.

Gameplay

The goal of our game was to design something similar to Crossy Roads but with a New Orleans twist. The goal of the game is the same as the original version of Crossy Roads: to make it to the other side of all the obstacles. However, to make the game less violent, we decided that when the pelican collided with the floats, it should stunt its progress by sending it downwards, but that it should not end the game. Additionally we included models that better fit the theme of New Orleans and Mardi Gras and animated the pelican to look like he is taking flight while moving up the screen.