WASHINGTON ROAD

Janet Lee, Ali Taylor, Phillip Yoon

Introduction

Washington Road is a Princeton-themed version of Crossy Road. The goal of this project is to enhance and customize a basic version of the game which randomly generates cars and trucks, creating a more engaging user experience. In particular, these changes largely fall into three categories: new terrain types with custom behavior, new obstacles which can kill the player or increase the player's score, and new animations and sounds for players to discover and explore. Our game would greatly benefit those in the Princeton community who can appreciate this tiger-specific version of the game.

Our implementation of Washington Road is based on a simple Crossy Road game which can be found here. The starter code organizes the game into lanes which can be of type car lane, truck lane, or a forest lane. The vehicle lanes generate a few vehicles into random positions. All vehicles in the same lane move with the same speed and in the same direction. Forest lanes have tree obstacles which players must navigate around, but there is no penalty if the player collides with them. Every time a player moves forward, a new lane is generated, and at each timestep, the code checks if a collision has been made with a deadly obstacle such as a truck or a car. Finally, the implementation uses a ThreeJS Orthographic Camera which follows the player as they traverse the road, but is otherwise stationary. While this project largely added new features to the Crossy Road game, there were a few notable changes such as altering the camera to scroll along the map over time, adding sounds and animations, and better integrating the score with the game.

The approach of this project is to improve and customize the simple Crossy Road game by applying and exploring a variety of graphics techniques using the ThreeJS library. We initially tried to implement the game from scratch, but realized that there was more value in significantly improving an existing simple version of the game. These improvements and customizations largely fall into three categories which include changing or adding a lane type, adding new forms of event handling, and improving the aesthetics of the game in novel ways such as adding death animations and sound effects which are not present in the original game. We used different approaches for each of these types of improvements, but all of our implementation methods depended on us looking up examples and ThreeJS tutorials online, or at previous COS 426 assignments for reference.

You can find our game at http://janetlee.me/washington_road/.

Methodology

Camera Movement: The starter code has a plain stationary camera that always follows the hero. However, we wanted to implement a constantly moving camera similar to the real game, such that if the hero takes too long to move and ends up falling behind the camera, we kill the hero. We implement this by enforcing that if the hero is off screen, they will be killed. For the camera, we have a steady speed in the y-direction of unit 1 per time step. To determine if the hero is in the viewport of the camera, we multiply camera.projectionMatrix by camera.matrixWorldInverse and set a frustum from the resulting matrix. Then if the frustum does not contain the hero's position, we kill the user.

Because the camera speed is often slower than the speed that the user is progressing, we also cause the camera to move faster if the hero is also moving faster. To determine approximately how well the hero is doing, we calculate the location of the hero in the 2D coordinates of the screen. If the y-coordinate of the hero in the screen is in the top 0.3 of the screen, we increase the camera speed to 8 units per time step. Otherwise, if the y-coordinate is in the top 0.4, we increase the camera speed to 4 units per time step, and if the y-coordinate is in the top 0.5, we increase the speed to 2 units per time step. This gave good results visually and the hero is almost always able to progress as fast as they wish to without having to wait for the camera to catch up.

Coin Minigame: The basic version of Crossy Road that we used as a baseline encouraged the user to avoid obstacles such as cars and trucks, and navigate around other world objects like trees in a forest, but lacked a goal for players. While players could increase their scores by staying alive, there were few incentives for players to take risks. The coin minigame is one way to encourage users to take risks to rapidly increase their scores. The coin minigame rewards players for collecting coins as they avoid obstacles and continue forward in the game. Each coin can only be collected once, and they are randomly generated.

The coins were implemented using the ThreeJS cylinder geometry and phong material type. As the player moves forward in the game, new lanes are spawned. Like the other objects in these lanes, the location of each coin is randomly generated. At each timestep, we check if the user's hero intersects with any part of the coin by taking into account the length of the coin and hero and checking for overlaps. If there is a collision, then the coin is removed from the scene, and the user's score increases.

Updated Hero Mesh: We created the hero to have the appearance of a Princeton student, donning a classic outfit of an orange shirt and jeans. The hero mesh in the original game was a simple chicken created from two box meshes grouped together; for our game, we created our own Mesh Group with arm, leg, body, and head elements to make the hero resemble more of a human, similar to a hero in Minecraft and in the real Crossy Road game.

New Terrain Types: We created a number of new terrain types for our game. First, we implemented a river with planks floating through it. The user's hero can only cross the river over the planks, and if the river carries the user's hero off the page, the user's hero dies. This terrain added an extra component of timing and urgency to move quickly, and generally adds more excitement to the game. For the planks, we added texture to make it appear more like a plank, and we also added texture to the river to make it more like a river. In order for the hero to be carried by the river, we had animate not only the planks, but also the hero if the hero is currently on the river. A difficulty in implementing river was that it allows for the hero to have a continuous position, whereas for the rest of the map, the user has a fixed column position. Therefore, moving off the river was difficult to implement and involved the hero's position being rounded to the nearest column.

To be original, we also created our own ice terrain. This terrain requires the hero to move off of the ice within 5 seconds of entering the ice. We have a cracking animation to indicate when the user dies from being on the ice too long; the ice texture changes from white to the blue water underneath it. In addition, to signal to the user that the ice will crack, there is an accompanying ice cracking noise that plays as soon as the user steps onto the lane. In order to keep track of the amount of time that the hero spends on the ice, we store the timestamp at which the hero entered the ice and when the current time exceeds that timestamp by at least 5 seconds, we kill the hero.

New Obstacles: To be original, we also added a different type of obstacle, a bagel ("Washington walk sign is on to cross -- bagels may not stop"). We created a bagel mesh using the THREE.TorusGeometry() method and chose to make it go back and forth on a single lane to distinguish it from the cars and trucks. We chose to create these obstacles because it changes direction and increases the activity occurring on the screen. We also imposed the UMatter and Public Safety textures onto the trucks by changing the materials on the truck and car meshes.

Sounds: We made the game more interesting by including sounds, namely, a real recording of Washington road's famous: "Washington, walk sign is on to cross. Washington, walk sign is... 8, 7, 6, 5, 4, 3, 2, 1, 0." This recording plays once at the beginning of the game upon initialization, using the THREE.AudioListener() and THREE.AudioLoader() methods. Another sound is a coin pickup sound that is triggered to play every time the user picks up a coin. Furthermore, in the ice lane, as soon as the user steps onto an ice lane, a sound clip of ice cracking plays for as long as the user is on the lane, including if they die after standing for more than 5 seconds.

Death Animations: While the original game does not have the chicken change its position when it dies, we implemented a death animation so the hero will fall on his face, either lengthwise or horizontally depending on the terrain. Due to the limitations of time for this project, we used the rotation() function for the Mesh Group of the hero to make this basic animation, but in the future, we can explore more complicated animations that involve individual body parts of the hero, such as flailing arms and legs.

Beginning and Ending Screens: We also added an introduction page for the game, explaining the rules and options for gameplay. In addition, we included an ending screen for when the player dies, which displays the final score and gives the user the option to play the game again. We used the same fonts as in Crossy Road to mimic the look and feel of the game, including Princeton Orange in our color scheme.

Game Difficulty: In order to make our game different than the original Crossy Road game, we also implement different difficulties. Therefore, from our starting screen, we have three difficulty play modes, triggered by pressing the 'E,' 'M,' and 'H' keys which correspond to easy, medium, and hard difficulty levels respectively. The harder modes have faster speeds for the obstacles and planks, which are again randomly generated. Also, as the game progresses, all the speeds increase by a small amount. Therefore, by playing more and more, the difficulty increases. We implemented this because we wanted to increase the user's engagement with the game and not let them get bored. Because the game, particularly the hard mode, becomes quite difficult, it allows users to be more drawn to winning and achieving a high score, perhaps even becoming addicted to it!

Procedural Generation: We used procedural generation to create more realistic game layouts. For example, ice lanes and river lanes are unlikely to be next to each other, and more likely to be next to another ice or river lane respectively. Similarly, to create long stretches of roads with vehicles (similar to the multiple lanes on Washington road), there are increased chances that a vehicle lane will be next to another vehicle lane than another lane such as a river or ice lane. Figure 1 shows the distribution of procedural generation probabilities for the easy difficulty. We create similar distributions for the medium and hard difficulties. Note that for the easy difficulty, there is a high chance of getting a forest lane, especially consecutively.

Figure 1. Probabilities used to procedurally generate a new lane based on the current lane of the hero.

Current Lane	**Vehicle**	**Forest**	**River**	**Ice**	**Bagel**
Vehicle	0.14	0.68	0.08	0.05	0.05
Forest	0.12	0.62	0.12	0.07	0.07
River	0.08	0.57	0.27	0.04	0.04
Ice	0.11	0.53	0.11	0.21	0.04
Bagel	0.10	0.67	0.09	0.02	0.12

Over time, the difficulty of the game should also increase. As a result, for each lane, the probability that the next lane would be a "safe" forest decreased by a small amount, and the probability of the other lanes scaled accordingly. Therefore, after passing almost 100 lanes, the probability of getting a "safe" forest had significantly decreased, and there would be more obstacle lanes than forest lanes.

Results

Metrics of Success: We measured success both quantitatively in our own testing and qualitatively by requesting feedback from real users. Quantitatively, we played the game ourselves many times to check for bugs; each time we added a new feature, we tested it thoroughly in different conditions (sound on/off, Mac users versus PC users, different window sizes, user playing with expected gameplay and with unexpected keystrokes). We noticed whenever there were bugs (flashing graphics, images not rendering properly, collisions being handled poorly, hero dying excessively, etc...) and corrected for them individually. Ultimately, we measure our success by determining how **bug-free** the game is and how **"smooth"** and **intuitive** gameplay is. Furthermore, we wanted to create a game that is **fun**, **aesthetically pleasing**, and **enjoyable** to play, almost to the point where it is addictive. We wanted to make a game that was relatively intuitive to play and did not require extensive instructions because it mimicked the real game Crossy Road in its simplicity. To determine how "fun" the game is, we asked real users for feedback.

Experiments: We sent out a survey to other Princeton students and asked them to answer on a scale of 1-10 how much fun they had on the game and how they would rate the appearance of the game. We additionally asked them to provide qualitative comments about their experience and their highest scores on the easy, medium and hard difficulty levels.

Data:

Figure 2. Participant responses to "How much fun did you have playing the game."

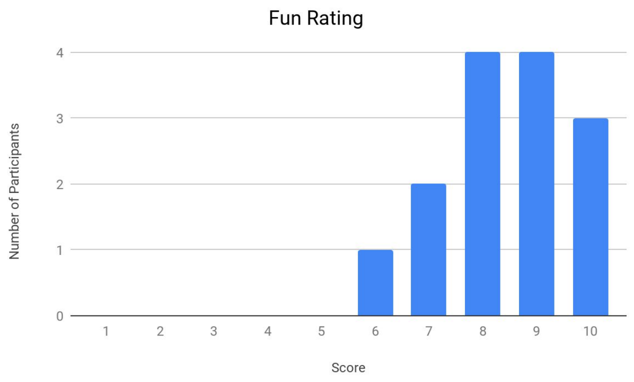


Figure 3. Participant responses to "How would you rate the appearance of the game."

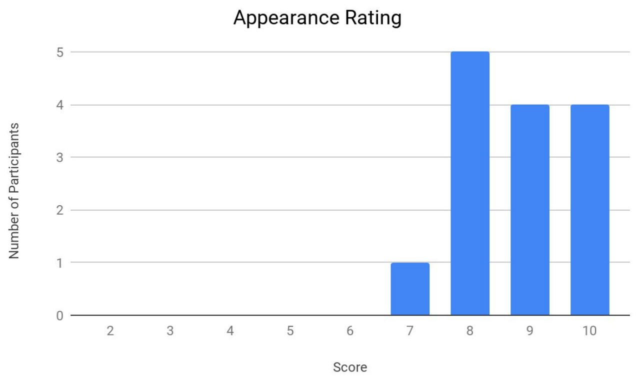
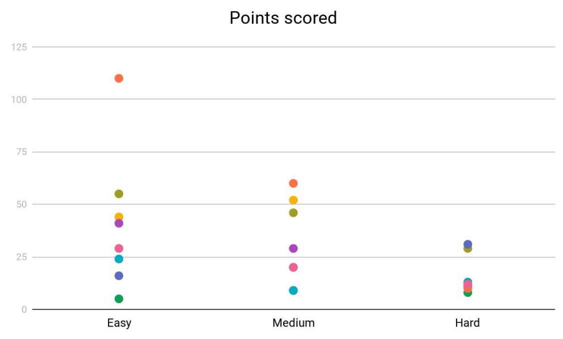


Figure 4. Participant scores on different difficulty levels.



Analysis:
Our results indicate that ultimately, our game met its goals of being fun and aesthetically pleasing. All users rated the game a score of at least 6 out of 10 for being fun, and the majority of users rated it at least 8 out of 10. All users rated the game's appearance with a score of at least 7 out of 10.

Figure 4 demonstrates that as the difficulty of our game increased, player scores generally decreased. This suggests that increasing the speed of obstacles at higher difficulty levels successfully increased the difficulty of the game. Fortunately, scores decreased in a reasonable manner, suggesting that the medium and hard difficulties were not too difficulty for players to navigate the game for the first few dozen lanes.

While most users reported that most of the game's mechanics were intuitive and understood what each lane represented, not all users realized that our bagel objects were meant to be avoided. As a result, there are additional steps we could have taken to make our game more intuitive.

Discussion

Overall, the approach that we took is promising because we were able to extend the original game from its basic features to a more exciting, dynamic, Princeton-specific game.

Other approaches besides our lane-based implementation include having a generic grid, with obstacles spawning with random probabilities at each point in the grid; this would allow obstacles to perhaps move across in the x- and y- directions, but our approach ultimately results in a game more similar to Crossy Road and is very scalable. We have a framework upon which we can continue to build many other features, such as more lanes (trains, lava, fire, etc.), obstacle objects (pickles, beagles, police cars, Eisgruber, etc.), sounds, animations, hero skins, and map theme colors. For the future, we can continue to gather feedback from users to determine what features they would like to see, and continue to make our game more intuitive by improving the graphics of the bagels to make them look more obviously menacing. Additionally, we can create more complex death animations and extend the coin minigame by allowing the user to collect coins over many sessions and win new hero skins as they play, just as in the real game Crossy Road. Furthermore, each skin can correspond to a new theme of the game, so a Princeton BSE

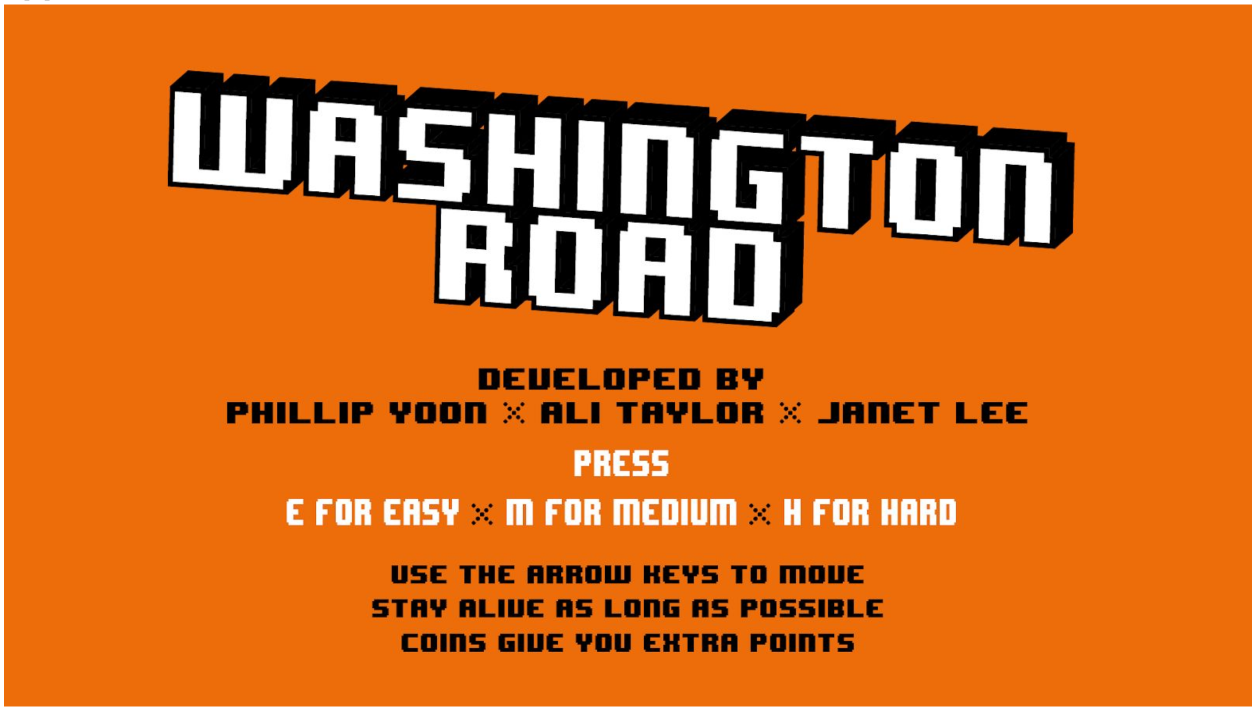
Student Skin can include lanes themed like the EQuad, while a Squirrel Skin can be themed to mimic the Wilcox/Wu area.

Through this project, we learned much about ThreeJS's documentation and the difficulty of game development, particularly the difficulty in creating meshes from scratch (in building the hero and other obstacles) and in creating dynamic animations.

Conclusion

This project successfully extended Crossy Road by adding new obstacles and features to make the game more fun and aesthetically pleasing for the average user. We used a variety of techniques to improve the original game which include modifying textures, procedural generation, modifying meshes, and adding audio listener objects. User testing demonstrated that most users found the game to be very fun and liked the appearance of the game. While there are other features that could be implemented as future steps such as more complicated death animations and hero skins, our current implementation of Washington Road provides users with an engaging experience they can taper to their preferred difficulty level.

Appendix: Game Screenshots

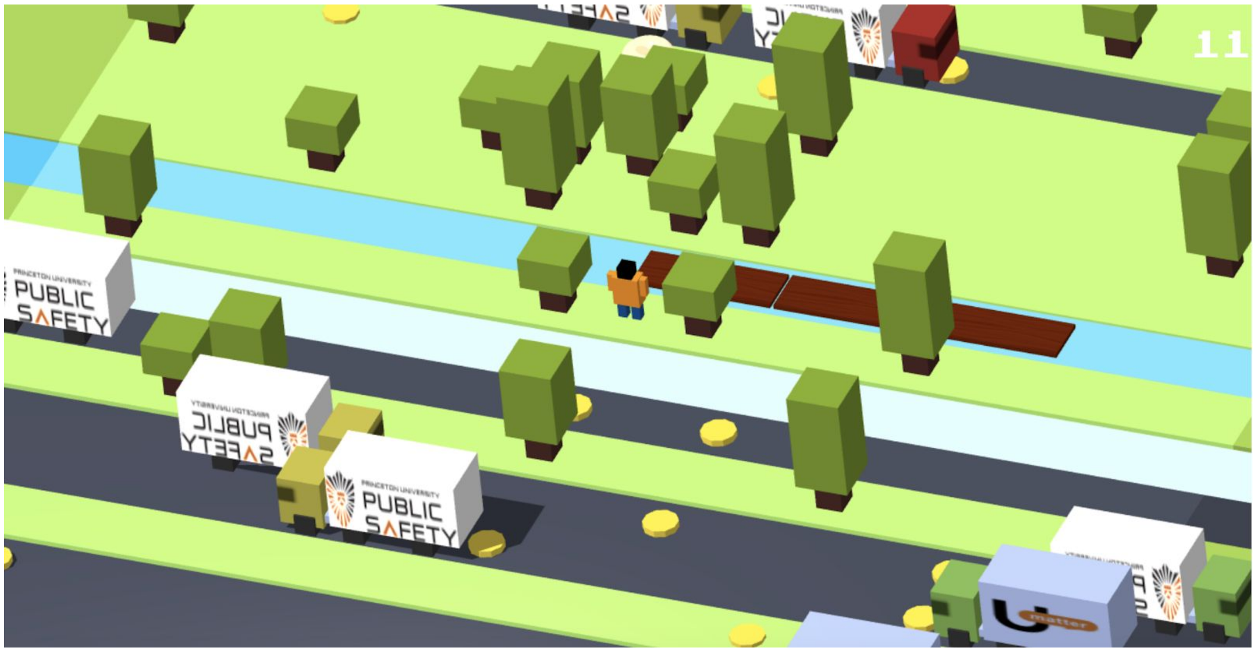


WASHINGTON ROAD

DEVELOPED BY
PHILLIP YOON ⁙ ALI TAYLOR ⁙ JANET LEE

PRESS
E FOR EASY ⁙ M FOR MEDIUM ⁙ H FOR HARD

USE THE ARROW KEYS TO MOVE
STAY ALIVE AS LONG AS POSSIBLE
COINS GIVE YOU EXTRA POINTS


11
PRINCETON UNIVERSITY
PUBLIC
SAFETY
PRINCETON UNIVERSITY
PUBLIC
SAFETY
U matter