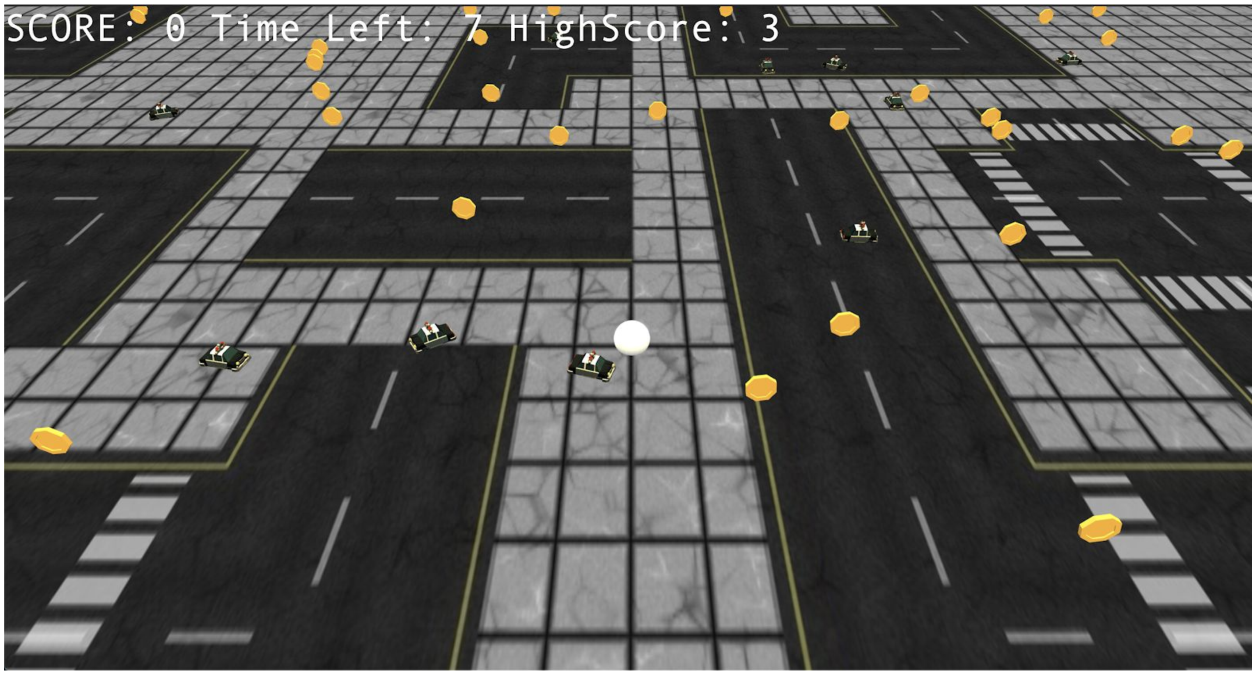
Crash for Cash

Final Written Report



Arman Medghalchi
Taylor Beckett

COS 426: Computer Graphics

Abstract

Crash for Cash is a single player game that operates with a simple objective, navigate around a map using your arrow keys, trying to collect as many coins as possible while avoiding police cars that are scattered throughout the scene. Starting with a welcome screen and ending with a game over screen, the user is engaged throughout the whole process, entering the scene as a stationary sphere that can be moved in any direction by the arrow keys. The physics is semi-realistic with an arcade-like feel, making the movement challenging yet responsive, while the camera follows the player from a distance and keeps the controls simple and reliable. We managed to successfully make an addicting arcade game that allows users to try to beat their high score while providing an engaging and reliable graphic landscape with room for further improvements and feature additions.

Introduction

For this project, we wanted to not only showcase our abilities to create an immersive environment, but to experiment with the basics of game-making, including making our own physics that would both allow us to custom-make our movement, and guarantee precise collision detection. We wanted to make a game that would be time-consuming and a little addicting. We think that anyone that enjoys arcade-type games and that enjoys striving to break their own records could spend hours competing with themselves.

One of the games that inspired our project was Cube Runner, a classic iOS game that we both played when we were younger and the iPod Touch and iPhone first came out. While Cube Runner was an inspiration, we altered the game to introduce our own take on it. We liked the aspects of Cube Runner that involved moving the player to avoid targets. However, we did not want to create an infinite run game as the market is flooded with these types of games.

There have been similar games, with the goal of either trying to move and hit targets, or to move and avoid enemies, but most of these games have a similar format, where the velocity is set and always moving forward, with the characters only options being to turn side to side. These games do a good job of keeping the user engaged while also providing something fast-paced. We wanted to keep this fast-paced and addicting nature while giving the user even more control over their movement. We wanted our game to feel a little arcadey and tough to control, as we believe that adds to the fun of it, with the risk and reward making the user come back for another game.

For our game, we decided to have a set 60 second game with a set number of randomly placed targets and enemies that reset each round. The player would be able to control their acceleration with each of the arrow keys, and the movement would be a little difficult to control, adding a layer of difficulty and excitement to each round. We think this should work well for people who enjoy racing against the clock and are competitive with their previous records, as this would allow them to continually replay the round trying to beat their score.

Methodology

Movement

There were several possible implementations for player movement. One of these possibilities was to follow the infinite runner approach, which involves a constant velocity with no noticeable acceleration changes, simply side to side or camera movement. Another possible implementation was to implement a super-realistic physics engine that would take into account every force present in the scene and generate realistic movement. Lastly, we could alternatively implement a very unrealistic and slidey movement, which would give the sphere an arcadey feel and keep the physics from getting too complicated, but might feel a little bit unnatural and therefore frustrate the players. We decided to not use the infinite runner implementation, as we wanted to provide the user more control over their speed and also implement features like acceleration.

From this point, we decided that we wanted to in a way combine the two extremes of movement physics. We implemented a system which would calculate acceleration with the vector direction calculated by the arrow keys pressed at each frame and the force being determined by a standard force that we determined experimentally. We also implemented a maximum speed, or terminal velocity, that the sphere was limited to, which was also determined experimentally and applied after friction was subtracted. For friction, we decided to simplify these calculations, with friction simply a percentage of the velocity that is subtracted from the current velocity. This allowed us to keep the physics calculations relatively simple, while also allowing for a slowing over time. This implementation for friction is relatively common, and we decided to set the friction variable to 30% of the current velocity, which gave us a semi-realistic look. This combination of features allowed us to curate the game movement to our wishes and create the semi-realistic, yet arcadey movement that we desired, with a proper standard force, maximum speed, and friction force to allow for responsive, yet somewhat challenging and fast-paced movement.

Scene Structure

There were many ways that we could have created the scene structure for this game. We decided to leave the scene relatively simple, with targets, enemies, a main player, and a ground texture. We decided on an engaging, yet relatively simple texture for the ground, and a very simple white rolling ball as our main character. For the targets and the enemies, we decided to pick coins and police cars. We imported both the coin and police car meshes from Google Poly in GLB files, and semi-randomly placed a certain amount of each on the map. We kept a grid variable when populating the coins to ensure that two coins were not right on top of each other, but aside from that, we wanted to ensure an element of randomness in each round that allows the user to get lucky with clusters of coins or slightly more difficult obstacles. This element of luck keeps the player engaged and wanting to continue playing.

For collision detection within the scene, we semi-randomly created x and z coordinates, translated the coin or car mesh by these coordinates and applied any rotations and scalings, computed the bounding box, and then set a bounding box state variable based on this translated and modified object. Once these variables were set, we simply had to update the position of the sphere based on the velocity, and detect whether the position of the sphere's bounding box has intersected the bounding box of any of the targets or enemies. We decided to keep the scene relatively simple, as we believe the interesting player movement and randomness of the objects will prove to be the main attractions and we did not want to overcrowd or over-complicate these aspects.

Camera

For the camera, there are certainly different strategies that offer different advantages and disadvantages. In approaching this decision, we considered several options. The first of which was having the camera in a single set position either slightly above the scene or directly overtop in a bird's eye view. The benefit for this camera approach is that it limits the movement of the scene and allows the user to focus on the movement of the player. In terms of the disadvantages of this approach, we felt that it limited the dynamics of the game and frankly, made the game boring. Another thing we did not like about the static camera approach is that it limited the size of the gameplay arena. We set the game play arena to a grid of 100 by 100 and with the static camera two things happened. With the camera set slightly above the scene, it was difficult to capture the entire scene in one camera view. With the bird's eye view approach, we would have to get so zoomed out that it was difficult to see the player and targets and enemies. The second approach was making the camera move with the movement of the player. We thought this had a much better effect on the gameplay. Not only could we have a larger scene for the player to explore, it also made the game more dynamic, introducing movement in the scene that made the game a little bit more challenging to play which we liked. In terms of drawbacks, it made the game a little bit slower to render and load. Overall, we thought the advantages of the moving camera outweighed the disadvantages and that is the approach we went with.

Multiple Scenes and Code Structure

This is a description of the structure of the codebase. Our game, in total, has three different scenes. To accomplish this, we edited the given source code to where the first program that was loaded by the webpack configuration was the main operation center for the rest of the code. The rest of the program, gameplay, scoreboard, etc. existed in several Javascript classes. For a more detailed explanation, on load of the page, app.js is called. This program initially creates a webpage, using Javascript to alter the DOM and CSS to style, that serves as the Crash for Cash homepage. It contains the title, a description of the game and how to play, and a start button. When the start button is clicked, our code creates a new Gameplay object from the gameplay class. The gameplay creates the scene, renders it on the webpage and populates in with all of the elements necessary to play the game. Once the timer in the game hits zero, the gameplay object is killed and the scene, along with all of its elements, is disposed of. Replacing the scene in the webpage is our Game Over page, created using HTML (through Javascript) and CSS. It contains the summary of the game including your score and the high score of that session (we did not have time to introduce a back end to our game so the high score is not the all time high score, but rather the highscore in the set of games you have played, assuming you play multiple times in a row). Finally, the last element of the Game Over page is the restart button, which, when clicked, performs the same set of actions as the menu page. There are multiple different ways to create homepages and game over pages such as creating them with pure HTML. We decided against this approach as our adopted approach allowed us to, first of all, write the entire game in Javascript, simplifying our tasks by not requiring us to switch programming languages. Also, it allowed us to form our game around Javascript classes. This was very helpful in allowing us to make our code very modular and easing the relationships between classes.

Results

We primarily evaluated our success on a set of two criteria: how fun the game is and how good it looks. The criteria, while subjective, gave us a good measuring stick to how well we performed. Our overarching goal was to create a game that looked really good while also being fun to play. In evaluating the game's "funness", we were very satisfied with the enjoyment the game delivered. In our opinion, the game is easy to learn, quick to play, and challenging enough to not be boring, but not too challenging that the player gets frustrated quickly. In terms of the visuals of the game, we were also satisfied. The meshes that the targets and enemies are made from look very good and are complemented by the road texture of the ground.

Satisfied with our regular goals, another way we evaluated success is how many of our reach goals from our initial project proposal we were able to accomplish. On this front, we did not achieve as much success. With only two people in the group, the best use of our time was to make the achievable goals be as robust and as polished as possible.

Discussion

Overall, we believe the approach that we took was very promising, and provides an entertaining and addicting game. We believe that the semi-realistic, yet arcade-like movements and the simple scene structure add to the experience, creating a game that is simple yet addicting. We wanted to avoid any overly complicated features that would only distract from the objective of the game. We gave the user complete control over the movement and acceleration, unlike infinite run games, and made the map and goal manageable, yet too big and obstacle-filled to complete in a 60 second round. We believe we do a good job of creating an engaging scenario and landscape, yet avoid overwhelming the player with tons of features that get tiresome and drive the player away.

We have several different follow-ups for this project, including adding more dynamic movement to the scene as well as making the graphical element more interesting. We could potentially add effects when a coin or car is hit, and add more movement to the objects. In terms of features, some easy additional features that could be easily added are a race against time feature, where you must collect all the coins as fast as possible, and adding various speeds and difficulties that the player can choose.

Throughout this project, we learned many very valuable lessons and methodologies. First, we learned that certain tasks will take a much longer time than first expected, especially when importing anything from online. We encountered this when we had difficulty importing and loading the coin and police car objects. Additionally, we learned the importance of fully understanding the meshes that you were altering. We ran into a lot of trouble with updating bounding boxes and updating geometries until we fully realized what part of the mesh we should be altering and checking. Finally, we realized the importance and difficulty of making tradeoffs. There is only so much time and so many resources available to us, and we had to decide as a group which directions to pursue and which tradeoffs we were willing to make.

Conclusion

As touched on in our results section, we successfully accomplished our base goals. Our product, Crash for Cash, is an enjoyable arcade style game that requires the player to navigate a scene full of cops while trying to collect as many coins as possible. We successfully introduce multiple meshes into the scene while smoothly handling collision detection, movement physics, and scene updating. However, we fell a bit short of our stretch goals that included changing the surface of the scene to be a 3D object instead of a flat scene and introducing Power Ups. These stretch goals would be the next steps. With a very viable base project to build off of, we believe given more time, we can introduce a great deal of interesting features. In terms of the issues we need to visit, there is a slight problem with lagging if a user plays several consecutive rounds. The performance issues can be slightly touched up and hopefully will be before the final presentations. Overall, we really enjoyed making this game, learned a lot, and are proud of our finished product.

Contributions

Taylor

For this project, I was in charge of creating the menu page, the game over page, and the connections that existed in between the different screens. Furthermore, I created the structure of our codebase. I also was in charge of several smaller pieces of the gameplay. In terms of the scene, I focused on the implementation of the camera of the scene and ensuring that the camera had the ability to follow the player. The creation of the scoreboard and the texture of the ground of the scene was also left under my control.

Arman

For this project, I was mainly focused on the gameplay aspect. This included importing, loading, and generating the objects and their positions, adding and connecting these objects to the scene, creating the movement and collision physics for the player sphere and the targets and enemies, and updating and connecting these elements to the reward and penalty system.

Works Cited

https://www.gamasutra.com/blogs/MarkVenturelli/20140821/223866/Game_Feel_Tips_II_Speed_Gravity_Friction.php -- For a reference on video game physics
https://opengameart.org/content/road-tile-textures -- For the ground texture
https://poly.google.com/view/0QjRoZvnOKT -- Coin 3D Mesh

https://poly.google.com/view/edxXN7Fah1m -- Police Car 3D Mesh