Alicia Liu, Cheyenne Zhang
COS426
Professor Felix Heide
10 May 2021

Final Project Written Report: The Cakery Bakery

Abstract:
The Cakery Bakery is a fun and interactive cake-making game where players rush against time to complete cake orders. The interface consists of plates passing by on a conveyor belt, and the player will have to drag the correct cake base, frosting flavor, and toppings onto it to construct cakes according to the current customer's order. There will only be a limited time before the plate disappears off the screen, so the player must be quick. In addition, the conveyor belt will speed up as the game progresses in difficulty. In terms of score, the player collects points for every correct order. The player also has a total of three lives, which will be lost if an incorrect cake is made or an order is missed. Visually, the game will have a retro 8-bit pixelated style.

Introduction:

We were initially inspired by two mini-games from our childhood: Club Penguin's *Pizzatron 3000* and Cool Math Game's *Papa's Cupcakearia*. The former is a conveyor belt mini-game where players build pizzas according to customer's orders under time constraints, and the options for pizza sauces and toppings increase as play progresses. The latter is a cupcake-decorating game, which is somewhat similar in the sense that the goal is to fulfill a customer's order, but there is no time constraint. We wanted to create a game similar to both of these, but with our own personal touches. More generally, we both played a lot of conveyor belt-type games when we were little, and thought it would be unique and interesting to create a game following this model, especially because we didn't see many similar games in the Hall of Fame of past COS 426 final projects. We also decided that we wanted our game to have a retro 8-bit pixelated style visually, and having an aesthetically appealing and cohesive theme became very important to our game. To accomplish this, we chose to hand-draw all of our graphics from scratch, rather than sourcing images online. This way, we could personalize all of the visual elements to our liking, and also ensure that the different pieces of the game would blend together well.

Methodology:

As mentioned above, visuals are a very important aspect of our game, and as such, we decided to hand-draw everything because it gave us more freedom to personalize the elements and maintain a cohesive theme. Initially, we did explore the option of sourcing meshes and images online, but it was difficult to find usable elements that fit the retro pixelated theme we were imagining. Even when we did find some, they didn't quite work

well in tandem with each other. Another factor was that we needed to have individual pieces of the cakes (bases, frosting, and toppings), and so even if we found an appropriate pixelated cake, for example, we would also need to manipulate it so that we had an image of the topping by itself or the frosting by itself. It was also important to us to make our game stand out visually and include our own personal touches in the game as well, even though this did mean we spent a significant amount of time designing and drawing the elements ourselves.

To draw each of these artifacts, we used Piskel, an online pixel-art editor. Its functionality is somewhat similar to Windows Paint, but is made specifically for pixel-art, which is exactly what we were looking for. We started first with our background. We searched online for ideas of what we wanted the background of our scene to look like, and decided on a color scheme that we would use throughout the pieces of the game. We came across a pixel-style image of a kitchen scene that we drew inspiration from for our game background. Ultimately, we added our own elements like shelves with the various cake parts and our own chosen color scheme (sunset colors, mainly peachy pink, orange, and yellow).

The final background image.

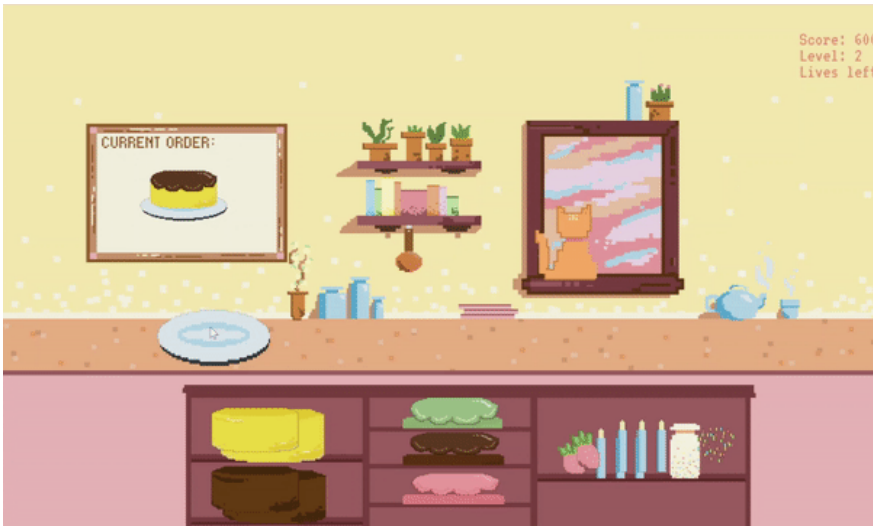
Aside from the background, we also had to design and draw all of the different cake combinations. We had decided on two cake bases (chocolate, vanilla), three frostings (chocolate, matcha, strawberry), and three toppings (candles, sprinkles, strawberry). This step was relatively time consuming because even with this relatively limited choice of cake parts, there are still a total of 18 different combinations. On top of this, we also

needed different files for each intermediary cake (i.e., just the plate, plate with just the cake, plate with the cake and frosting, and the total completed cake with cake, frosting, and toppings). Aside from the in-game elements, we also hand-drew the favicon and on-screen titles. The favicon was drawn on Piskel, and our title screens (start, instructions, controls, pause, game over) were made of backgrounds drawn on Piskel, and then we overlaid pixelated text onto it using the popular and easy-to-use graphic design platform Canva.

Once we created the drawings, we focused on the game itself. The game is built using three.js. This choice was partly because we were familiar with this library from previous assignments and also because there are a variety of built-in features that lend themselves well to the gameplay that we were designing. For one, we were able to utilize three.js's DragControls to facilitate our dragging movement, i.e. dragging cake elements onto the plate. This was an important part of our game. To turn our drawings into objects in the game, we used three.js sprites. Sprites are planes that always face the camera, and sprite textures can be easily imported from an image. This was the best way to create our objects because the visual style we were going for, i.e. 8-bit and pixelated, meant that our game would be designed in 2D. It wasn't particularly necessary for us to depth-of-field or different perspectives for the objects in our game.

Another important aspect of our game is the auditory stimuli. In order to create an immersive experience, we had to look for sound effects and background music. We did not really have enough music background for us to create this ourselves, but thankfully, it is easier for sound effects from different sources to match thematically than for visuals to. We had two main sources for sound effects: *8-Bit Sound Effects Library* by LittleRobotSoundFactory on FreeSound and *The Essential Retro Video Game Sound Effects Collection* by Juhani Junkala on OpenGameArt.

We will now jump into a walkthrough of gameplay. The game starts with a start screen overlaid on a 50% opaque background. We used event handlers to handle key presses as the user moves between title screens. Key presses also play an important role in the game itself. Users can see instructions and controls using the "I" and "C" buttons respectively before pressing the spacebar to start the game.



Example of gameplay: making a cake according to the order and submitting it.

A plate moves from left to right on the screen along a conveyor belt. This is done by saving the plate in an update list in our scene (we will describe our scene more in detail below), and calling an update function on it every 0.5 seconds, which increases the x-position periodically. The speed of our plate varies, so it is passed in as a parameter. The default is 4 pixels per 0.5 seconds. Then, a player sees a picture of the desired cake, which is the current order, on a board on the left side of the screen. The player then must drag the correct components onto the plate and let go. If it is correct, a correct sound is played and the component will "snap" into place onto the plate. This is done by checking the position of the dragged object and of the moving plate; if they are within a certain distance from each other, then we replace the texture on the plate sprite with the image of the plate and the newly added object. We only allow adding cake bases to a plate, frostings to a plate with a cake base, and toppings to a plate with a cake base and frosting. If it is done in the incorrect order, an error sound is played and the dragged object returns to its starting position on the shelf. The player presses the "S" key to submit the order (or, if the plate reaches the end of the conveyor belt, the current arrangement is automatically submitted). The order board shows a green checkmark if correct and plays a correct sound and a red X if incorrect and plays an incorrect sound. The plate then speeds up (i.e. the step size passed into the update function is increased to 50) and disappears off the right side of the screen. Once it does, it reappears, empty, on the left side, and a new random order is generated and its picture is placed on the screen. The player continues to play and will level up every so often. Level 1 is only cake bases, level 2 is cake bases and frostings, and level 3 is cake bases, frostings, and toppings. The speed of the plate also increases within the levels. The player has a total of three lives, and every time an incorrect order is submitted, a life is lost. When the player runs out of lives, the game is over; a game over screen appears and the player can choose

to press the spacebar to play again. The score, level, and lives are displayed in the top right corner of the screen.

We will dive deeper into the specifics of the code. Most of our game logic is housed in the main file, app.js. The state of play has four options: NOT_STARTED, PAUSED, PLAYING, and GAME_OVER. These states inform what key presses and controls are available to the player. For example, in NOT_STARTED, PAUSED, and GAME_OVER, the background and everything in it is changed to an opacity of 50%. The player can also press "I" and "C" to see the instructions and controls respectively when not in play.

For the scene, our game consists of one scene in KitchenScene.js. In the KitchenScene constructor, we load a sprite for our background and add variables in the state that allow us to parse important information about the current gameplay. More specifically, we store what needs to be updated at every timestep (the plate and whatever is on it), an array for all the draggable objects in the scene, an array to save the elements on the current plate in the order they were added, a variable to store the current customer order, which is represented as an array with the first element being the base option, the second element representing the frosting, and the third element representing the topping, as well as booleans to indicate if the plate is at the end of the conveyer belt and if the order has been submitted. Within the KitchenScene, we also declare functions to update the scene at every timestep, add and replenish the ingredients, add and clear orders, and display the welcome and instruction panels over the scene when the game is initially loaded.

To detail some challenges of the process, it took a fair amount of time to actually get started. Our idea for the game is quite different from a lot of the coding we've done in the class so far, so it took some experimentation for us to get our setup going and understand how to build up the game the way that we wanted. It was helpful for us to look through the source code of previous Hall of Fame projects; specifically, the Driver's Ed game from Spring 2020 was very helpful to understand everything from basics like setting up a scene to more specific features like playing audio. One other somewhat specific tricky aspect was figuring out how to automatically submit an order once it reaches the end of the conveyor belt. We were previously only using a boolean variable *submitted* to check whether or not it would be necessary to submit. In the end, we decided to add a boolean variable *atEnd* which would be set to true in the scene's update function once the position of the plate was outside of the boundary, and to check this in the return from update and call *submitOrder()* as necessary. We also decided to pass in a step size to *submitOrder()* so that the step size following the submission would make sense (fast if still moving to the rightside, slow if starting from the left again).

Results:

Currently, we have the majority of our features for our minimum viable product (MVP) completed. Our Kitchen Scene renders successfully, along with the welcome and instruction panels overlaid on top when the page initially loads. We have implemented the basic gameplay features, including having randomly generated customer's orders displayed on the menu and having plates move along the conveyor belt, onto which players can drag cake bases, different frosting flavors, and different toppings. We also have a scoring and lives system, so users earn points for each cake they make correctly and advance to new levels when their points reach a certain threshold. We currently have three levels implemented, with each level getting progressively more difficult, as the cake orders get more complicated and the plate moves faster so players have less time to complete each order. The player has a total of three lives and the game ends after the loss of the third. Finally, we also implemented sound effects for the main events that occur in the game, including starting, completing an order correctly and incorrectly, and ending the game.

Once we completed our main features, we also had some of our peers try out the game and give us user feedback. Specifically, we prompted them about what they liked about the game as well as areas they thought may need improvement. Overall, we received pretty positive feedback, with one player even saying, "overall very immersive experience, it really felt like I was in a bakery." Our peers generally enjoyed the graphics and overall aesthetic of the game, although there were certain areas of the gameplay they thought could be improved. More specifically, one person thought the instructions and controls were a bit hard to remember, and a few people also thought the speed of the conveyor belt needed to be adjusted, as they felt it was too slow at the beginning and too fast at the end. We will be spending time in the next few days modifying our game according to the user feedback that we received.

Discussion and Conclusion:

Overall, our approach has been effective and we have been successful in implementing most of the features that we wanted and achieving the goals we set out for ourselves. We have received positive feedback from users and seen them really enjoying the game that we've created thus far. Since we both came into this project with very little experience with three.js outside of this class and with little experience designing and creating games, we had to pick up the necessary skills and knowledge pretty quickly by watching videos and reading tutorials online in order to keep up with the project's timeline. Through this process, we've definitely learned a lot about the mechanics underlying games, as well as how we can integrate our own personalized elements and designs into the gameplay.

Moving forward, we first want to work on a few bugs that we have. One main bug that we need to fix is that our objects, including the background, are displayed differently on

various devices because of differing dimensions on screens. Additionally, we want to spend time before the demo improving our game according to the user feedback that we received, as mentioned above, as well as work on hitting some of our stretch goal features. Depending on the amount of time that we have left, we want to work on adding additional animations to the game, for example having movement in the background or animating the process of adding frosting and toppings, as well as implementing a backend for our game so we can store and display a leaderboard of top scores from players. These stretch goal features would definitely help to enhance the user experience during gameplay, and allow the current player to feel like they are competing with other players and give them extra motivation to advance further.

Contributions:
Initially, Alicia worked on designing and drawing the artifacts of the game, while Cheyenne worked on the game logic and setting up the code structure. We made major decisions about gameplay together, such as scoring and levels. We worked together on the proposal presentation, written report, and will work together on the demo as well.

Works Cited:

- https://threejs.org/examples/
- http://www.threejsgames.com/extensions/
- https://freefrontend.com/three-js-games/
- https://twitter.com/scrixels/status/1174676914990178304
- https://codepen.io/HunorMarton/pen/ExNzWqm
- https://www.youtube.com/watch?v=JhgBwJn1bQw
- https://www.youtube.com/watch?v=FwcXultcBl4
- https://observablehq.com/@grantcuster/understanding-scale-and-the-three-js-perspective-camera
- https://github.com/mrdoob/three.js/blob/master/examples/misc_controls_drag.html
- https://www.youtube.com/watch?v=gEZcJ3GufmE
- https://threejs.org/examples/?q=controls#misc_controls_drag
- https://github.com/karenying/drivers-ed/blob/master/src/components/objects/Bus/Bus.js
- https://github.com/jimmyyhwu/shape-battles/tree/master/public/js
- https://freesound.org/people/LittleRobotSoundFactory/packs/16681/
- https://opengameart.org/content/512-sound-effects-8-bit-style