Cathy Teng, Colin Reilly, Matthew Trotter
COS 426
Professor Felix Heide
3 May 2022

Byte Heist

Abstract

Byte Heist is a 3D platform ball-rolling game where the player is a computer virus and the goal is to collect 8 bits (or a byte) from each level before the timer runs out. The timer indicates the time left to play before the virus is "caught" by the computer's antivirus. We include three types of obstacles in this game that are based off of circuit board components – resistors, capacitors, and copper wires. Resistors are essentially small platforms that the player can jump on, capacitors bounce the player back, and copper wires shock the player and cause them to lose the game. There are 3 levels of varying difficulty as well as pause and restart functions. This game makes use of the Three.js library for building 3D computer graphics and Cannon-es.js for the physics functionalities.

Introduction

3D platform ball-rolling games are not new – in 2002, Marble Blast Gold was released and was a big part of our childhoods. While the basic premise has been around for a while, the benefit of this type of game is the multitude of themes that can be applied to make the whole project original. We were interested in making a marble rolling game with a computer-y theme, where the "ball" is a virus, and the platforms are computer chips or circuit boards (think the classic green boards). Whereas Marble Blast Gold has a leaderboard with the fastest times and the goal is to collect the gems and make it to the finish platform, Byte Heist features a countdown timer and the goal is to collect the bits and make it to the finish platform. Once the time runs out, the player is "caught" by the antivirus and they lose the game. Marble Blast Gold also includes various obstacles and power ups, such as slippery mud and jump boosts, but in the time span we had to implement the project, we decided to go with a few obstacles, and one can also be used as a boost if you can figure it out. We emulated three different obstacles in the form of resistors, capacitors, and copper wires. The resistors and capacitors are custom meshes that we downloaded from the Internet and colored ourselves. We also wanted to make it level-based to simulate different parts of the computer and ended up designing three levels of varying difficulty. We were inspired by a few COS 426 Hall of Fame Final Projects, most notably Going Viral, which coincidentally also has a "virus" theme (albeit within the human body). The premise of the project is also versatile in that we can continuously add more features (obstacles, levels, powerups) without changing any of the underlying structure after completing the basic features. Overall, it is a simple game with a simple and fun plot line!

Approach

Our main idea was to make a game in which the user, as a computer virus, would roll around collecting ("corrupting") bits on a computer chip to deliver them to an end platform within a certain time frame to beat the level. This would include navigating the spaces and platforms available within the level as well as avoiding or using the obstacles to their advantage. This allows for some interaction between the user and the game, and we tried to design the obstacles such that their interaction with the virus makes some sense in the real world. The game

is expandable in that there are levels that we personally designed, but realistically it shouldn't have too many because a computer doesn't have that many chips or boards.

We were largely inspired by the games Marble Blast Gold and Going Viral, the latter of which is featured in the COS 426 Hall of Fame. We based our controls on the controls in Going Viral – that is, the left and right buttons don't move the player left and right but rather shift the camera angle to the left and right. We were also introduced to Javascript physics handlers and utilized *Cannon-es*, which allowed us to focus more on asset development, realistic ball rolling, some aesthetics, as well as prioritize more time for level development and creation. While Marble Blast Gold doesn't have a specific theme besides rolling around as a marble and collecting gems, including a fun computer-y theme in ours will hopefully attract and entertain more people.

We considered having enemies in our game, like something representing firewalls, but decided that since the inside of a computer is generally pretty static, random enemies didn't really make sense for our game. This idea was spurred on by the virus enemies in Going Viral. Instead, we spent more time designing and implementing levels, which are much more complicated than the straight lines in Going Viral. It also didn't make sense to have the same straight platform for all of our levels since different computer parts are also of different sizes, and we don't have bounding walls because, of course, computer chips and boards don't have those. We do enjoy straight runner games, but we wanted a more exciting environment and thus strayed more towards a platformer with some puzzle-like qualities as well as a speed component.

We worked to create a level-based game as opposed to infinite generation, created assets that make the world more interesting and explorable, and created obstacles that provide some difficulty to the levels. We opted to download pre-made assets and color them to our liking to include the resistor and capacitor obstacles that we wanted .We aimed to create interesting, varied levels in difficulty and obstacle arrangement. Lastly, we attempted to encourage replayability and induce a (non-stressful) sense of urgency by providing a timer that the player can use to improve their completion time in subsequent rounds, as well as a final timer of all time taken to complete all levels. Our "computer chip" theme evolved in the level design phase into more of an inspiration rather than a strict theme, since a computer chip doesn't usually have resistors and capacitors, at least not in the arrangement that we present them in. We recognize that it isn't entirely accurate, but we hope that the decision to use an entertaining theme loosely over an entirely accurate one does not cloud the fun a game like this could provide.

The way the game starts and ends is also in line with the theme. The player is a virus that is "dropped" into the world and must roll around platforms representing computer chips and collect floating bits to win. "Winning" means that the virus succeeded in collecting bits and messing with the computer before getting caught by the antivirus, so the "game won" screen is glitched. The game might be less replayable than an infinite generation game because we force the player to start from the beginning if they lose and the levels are always the same, but it is still easily scalable. Infinite generation with 3D assets was difficult to implement because of having to keep track of all of the assets to be able to simulate collisions with them. It probably would have also slowed down the game because of having to constantly loop through all the obstacles. Thus, we keep the game exciting by introducing time and collection requirements.

Methodology

In order to implement this approach, we had to implement assets and obstacles, menus and general game tools, levels, and virus movement.

Assets and obstacles:

Many of the obstacles were via assets that were downloaded online, although the walls, platforms, bits, and copper wires were made with THREE.js meshes. As far as assets go, none of us are experts in Blender, and thus figured that we would make use of readily available resources whenever possible, though we still would edit their appearance as needed. We obtained the resistor and capacitor assets from GrabCAD, from users *singlefonts* and Alex Fedorov, respectively. In addition, we were able to change the 'texture' of the walls and floors using the ThreeJS materials constructor to load in custom skins. If we were to make the assets ourselves, the quality of them may have been lower, as we haven't had the experience. However, we still did customize the colors of the resistor and the capacitors and created the virus and bit objects ourselves using Blender. In order to work with the cannon-es physics handler, objects must be imported into the ThreeJS scene as well as represented in the Cannon world to calculate collisions. We created the bounding boxes around the resistor and capacitor by hand, and opted not to create bounding boxes around the bit and copper wires and calculated the collisions ourselves. All other objects were easily transformed into Cannon bodies, and we could arrange these in non-trivial ways to make levels that are entertaining to play and can efficiently run. The collisions with the 3 obstacles we created are as follows: you can jump on resistors, bounce off of capacitors (on all sides), and will lose the game if you touch copper.

Menus & general game tools:

We attempted to package the different assets within the code such that levels could be built easily and efficiently. We specifically tried to include a similar placement and shape of the meshes and Cannon colliders of the floor across levels as well as repeated patterns of obstacles within levels. This required a large amount of repetitive hard-coding, especially with determining exact positions of things and because each object has to be initialized separately.

However, despite the attempt at efficiency, there is still a period of time that it takes to load the meshes in for each level and conduct the under-the-hood workings for the game to run. We had to accept that there would be a loading period, as the asynchronicity of the loading of the meshes prevented their reference at compile-time, causing issues in pre-loading. Ideally, we would have loaded each pre-made asset once and simply cloned it over and over. We got through this by adding a loading screen via CSS that allows a few seconds to transpire before showing the level. We also thought it might be helpful to not only pass the time but help the player by giving some hints on the loading screen.

We include a click to start/resume and a press Esc to stop functionality by utilizing ThreeJS's PointerLockControls. We had some fun spicing up the CSS on the various screens in the game – the start and win screens have some glitchy text, and the game over screen features some glowing text that cracks. While the player is playing, the game stats (current level, time remaining, bits collected) are displayed on the top left of the screen and resemble a computer bash. The game animation and time remaining pause while the player is on the pause screen. When there are 5 seconds remaining in the game, a flashing red screen appears that counts down the remaining amount of time. If the player loses, they can click to restart the game which then loads in level 0. If the player wins, the win screen will display along with how much time they took to play through the whole game so they can compare with their friends.

Levels:

The levels were implemented such that each level is pre-made in its own function, and changing levels just requires re-initializing the scene, the world, and the objects in the scene/world. We designed the levels on paper and then hard-coded them into their respective

functions. For each level, there is a pre-specified amount of time that the player has to complete the level. They must traverse through the platforms and obstacles, collect 8 bits, and make it to the end platform (which will appear white and have an arrow pointing down towards it) before the time runs out. Falling off of the world will respawn the player in the same level but at the position in which they started, and losing will force the player to start from level 0.

We kept all the platforms on the same y value in order to keep the "falling off" state consistent throughout the level. We added the ability to rotate the obstacles to allow for some variation in how the level is arranged. As mentioned before, loading in all the obstacles, particularly for level 2, is slow and less than ideal because of the asynchronous importing of the assets. This could have been done better, but modularizing the level initialization outside of app.js was cleaner and worked well until we put a lot of resistors in level 2. Once the assets were added, it was not difficult to calculate collisions using a combination of our own code and the cannon-es physics engine. We debugged the level creation and object imports with cannon-es-debugger.

Virus movement:

In being inspired by Marble Blast Gold, we really wanted to be able to use WASD + spacebar as forward-left-back-right + jump controls and to be able to use the mouse to look around the world. Unfortunately, the various controls available in ThreeJS currently only support a first-person worldview while we use third-person (because you are looking at the virus moving). We borrowed the controls from Going Viral and ColoRing, where "moving" left and right actually just pans the camera left and right and moving forward and backward work normally. We use WASD for this and also the spacebar to jump. When going forward or backward, we apply a scaled impulse to the Cannon body of the virus, which also causes the ThreeJS mesh to rotate because we bind the mesh's position and quaternion to the Cannon body in the animation loop. We allow jumping when the virus's y-velocity is 0 + an EPS – this was simpler than trying to calculate if the virus was actually touching an object or the ground.

Music

Using royalty free music files downloaded from pixabay.com, the functions for each level is accompanied by a javascript audio object that plays during the level. The music plays during the loading screen and during game play and pauses when the game is paused.

Improvements:

There were several different things that we ended up not implementing. Given more time, we would ensure that the levels are in-depth and make our best attempt to have them be as architecturally accurate as possible. We could do this by connecting the computer components (our assets) via wire-colored connections in the texture of the floor. We do have copper wires, but again, the visuals aren't entirely accurate. We could also make the environment look more like the inside of a computer, whether it be through lighting or custom textures. Adding additional assets like magnets or inductors that propel you, a battery that changes your speed, etc. might make the levels more interactive and the computer chip theme more evident. Even though the visuals could be vastly improved, we hope that the theme comes across in our game's current iteration. At one point we considered having a boss fight at the end with the antivirus, but this wasn't implemented as we figured it wasn't as fitting with our theme and with the game's structure as a 3D platformer. Lastly, we recognize the camera is sometimes covered by walls, especially in the second level. Adding some type of backwards ray-tracing element to align the camera on that positions the camera at the closest intersection or max distance would prevent the virus from being blocked from view.

Discussion

We believe this project is promising, particularly in its scalability and unique theme. We personally find the game fun and also non-trivial; while there is room for improvement, we worked hard on the level design and are proud of the final outcome. Our choice to implement levels as opposed to an infinite runner allows for more customization in the look of the world. With an infinite runner you are always going in a straight line, which doesn't leave much room for fun interactions with the environment. Having levels was also more straightforward than procedural generation with 3D assets.

The most important follow-up would be to create more levels. As is, the game can be finished in a few minutes, despite the many hours we spent making it. We could do this by making levels that are more intense, with less time, or more obstacles, or levels that incorporate more creative usages of our obstacles and such. For example, we've found that the capacitors allow you to jump higher and higher the longer you stay atop them. This could be used to make a level where you are forced to reach a higher place, or a platform that's too far to jump to without a boost.

By doing this project, we've learned to code better in Javascript, especially with threeJS and cannon-es. We've learned to structure our code in a way that is modular while being flexible and making changes along the way, such as when we had to take into account asynchronicity with GLTFLoaders and decided to add a loading screen. Additionally, some of us haven't worked in a group like this working on a larger-scale generative and creative computer science project, and this has provided us the opportunity to work with others in mind, to work with others' coding styles, and to work with making our own code readable and understandable.

Conclusion

We believe we effectively reached our goal by making a game with at least 3 levels that scale in complexity, by including obstacles that you can interact with in interesting ways, and by providing a world that you can move around in to explore and collect things. While there is room for improvement, further changes can be easily implemented and what we have now can essentially scale infinitely. The theme is fun and unique, and people we have asked to play the game have also found the premise intriguing. The next steps would be to lengthen the game and make it more fun by increasing the number of levels, increasing the difficulty, and increasing the complexity of the interactions with the world like by adding new assets, or by using the existing ones in interesting ways. We should definitely revisit the manner in which we load the assets – it would be much faster if a .gtlf that is used repeatedly is loaded once rather than the number of times the object appears in the scene. Currently, the slowness in generating the levels sometimes causes the loading screen to animate a bit choppily. Overall, though, we are proud of what we have achieved in the short amount of time allotted for us to complete this project and hope you will enjoy it as well.

Contribution Breakdown

Cathy

- Organization
  - Created modularity in the code (initialization and levels), organized tasks for the team to divide-and-conquer, debugging and code clean up
- Camera & movement
  - Borrowed Going Viral's basic controls and camera movement, added PointerLockControls
- Game structure
  - Point person for game logic, created the architecture for setting levels
- Menus
  - Created screen menus and stats with JQuery and CSS
- Assets
  - Colored resistor and capacitor in Blender, created Virus object, created Bits with Colin

Colin

- Level design & implementation
  - Designed three levels utilizing:
    - Resistor objects
    - Capacitor objects
    - Copper objects
    - Bit objects
    - Ground planes (Cannon, three)
- Jumping
  - In order to prevent jumping while in the air to perform a "double jump" or "X-time jump" the velocity in the y direction was checked when the spacebar was pressed and if it was less than the threshold the jump force was applied if not there was no force applied.
- Audio
  - Using pixabay.com, downloaded royalty free music for level audio that changes by level and pauses on the pause screen
- Start End Platforms
  - Created platforms for the virus to spawn on (Start platform) and a platform for the level to end on that generated a signal to indicate the user has completed the level (end platform)
- Texture
  - Using google images, found images to use as skins for the ground and walls of the levels.

Matt

- Theme and idea generation
- Assets
  - Obstacle idea generation and consequential world interaction
  - Finding the models
  - Coding the helper classes and events that coincide with them

    - Resistors: collision detection, ensuring that (axis-aligned) rotation works on the several Cannon colliders that contribute to the object, allowing movement on top of an under the resistor too.
    - Capacitor: Repelling, the virus gets blasted back by a jolt of energy. The model had to be reshaped and edited, as well as new materials and material interaction defined.
    - Bit: The collection of the bit based off of distance to the virus and subsequent counting of the bits collected to reach an end game state.
    - Wire: Upon entering the bounding box, the game state is changed to lose the game.
- Writing

Works Cited

Three.js, a library for creating 3D graphics, manual available at https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene.

Cannon-es.js, a maintained fork of physics engine Cannon.js, by Stefan Hedman, https://github.com/schteppe/cannon.js.

Assets

- Resistor Carbon Film 0W25 L7 D2.5 LD0.65mm Horizontal.stp from Resistor Carbon Film 0W25 | 3D CAD Model Library | GrabCAD, by user *singlefonts*, https://grabcad.com/library/resistor-carbon-film-0w25-1.
- CAP_BOX_7.3x3.5x8_BL.stp from passive THD | 3D CAD Model Library | GrabCAD, by user *Alex Fedorov*, https://grabcad.com/library/passive-thd-1.
- arrow.obj by user *darkdrew*, https://clara.io/view/b8a8fb8d-cfe7-4e5d-a219-e26f862feb42
- Circuit board texture https://i.stack.imgur.com/VYKhJ.jpg
- Firewall texture https://media.istockphoto.com/vectors/matrix-background-vector-id1340505444?k=20&m=1340505444&s=612x612&w=0&h=GAu97ih58f7pmSYlO3aYZTQEwMziGwjYqw1onzeBDVk=

Game design / organization / controls:

- Going Viral by Jeongmin (JM) Cho, Claire Guthrie, Heidi Kim, Oliver Schwartz, https://github.com/oliverschwartz/going-viral
- Get Down by Zoe Kahana, Willie Chang, Khatna Bold, https://github.com/khatna/getdown

CSS

- Black Mirror Cracked Text Effect by user *George W. Park*, https://codepen.io/GeorgePark/pen/jeBbGN
- “How to create "Blinking" text with CSS only?” from https://stackoverflow.com/questions/43720936/how-to-create-blinking-text-with-css-only answered by user *Niels*, https://jsfiddle.net/yo5kw3qd/
- Animated Charging Border (@property) by user *Jhey*, https://codepen.io/jh3y/pen/jOVNOeg
- Cyberpunk-Style Glitch Walkthrough by user *Matt Gross*, https://codepen.io/mattgrosswork/pen/VwprebG

Blender

- “How to create 3D Text in Blender” by user *Derek Elliot*, https://www.youtube.com/watch?v=cS3aRmcohn8
- “How to make a spiked sphere” answered by user *Paul Gonet*, https://blender.stackexchange.com/questions/56535/how-to-make-a-spiked-sphere

Starter code

- https://www.cs.princeton.edu/courses/archive/spring22/cos426/zips/COS-426-Final-Project.zip
- “Cannon.js Tutorial For Beginners - Add Gravity, Collision, And Other Physics Laws To Your 3D Web App” by user *Wael Yasmina*, code https://github.com/WaelYasmina/cannontutorial/blob/main/src/js/scripts.js