Princeton Run: a day in the life of a Princeton Student

1. Abstract

In our final project for COS426, we developed *Princeton Run*, a fast-paced game inspired by the mechanics of *Temple Run*.[1] Built using JavaScript and the THREE.js library, the game puts players in the shoes of a Princeton student navigating the streets in pursuit of the highest possible score. The player's goal is to run for as long as possible, avoiding obstacles by strategically moving to the side or jumping over them, all while collecting coffees and letter grades. We leveraged the provided course repository as a foundation and built *Princeton Run* on top of it, supplementing it with GLB models sourced online (see below for list of created vs. sourced). A key feature of the game is its procedurally generated road system, which dynamically extends ahead of the player and removes segments behind them, creating an efficient infinite-runner experience. The most challenging aspect of the development was implementing smooth turns and the correct orientation of the player. Finally, we implemented a leaderboard that allows all players to compete with each other and adds a competitive nature to the game.

2. Introduction

Princeton Run is an endless runner game that mixes Princeton-themed elements with the gameplay concepts of the famous single-player game *Temple Run*. The player represents a Princeton student who has to navigate a road filled with Princeton themed obstacles, grades, and coffees. It represents how many students at Princeton feel: a race to infinity with the sole purpose of not crashing into walls and obstacles, following the honor code, and simultaneously picking up as many good grades and coffees as possible. The goal is simple: survive as long as possible. Developed using JavaScript and THREE.js, *Princeton Run* showcases the techniques and tools explored throughout the semester in COS426.

The game is built on top of the course-provided repository, which served as the foundation for our implementation. While originally we wanted to build around a different repository, we noticed that the COS426 repository we set up was much more suitable to expand and build upon. Our key features include the dynamic generation of road segments, including turns, jumps and falls in road height, seamless deletion of outdated parts, the integration of GLB models and audio to enrich the game environment, object collision detection, plus several different metrics for success. We created custom assets for grades and the Rights, Rules and Responsibility book which features an important role in the game. The player's experience is enhanced by the ability to jump and turn as can be expected from similar games. In addition, collecting 7 coffees allows the player to sprint without hitting obstacles, analogous to *Temple Run*'s Power Up mode achieved by reaching enough coins. The *Princeton Run* player runs in one of 5 lanes, simplifying the game experience and development. Implementing smooth turns, jumps and aligning the player's orientation to these transitions proved to be the most technically demanding aspect of the project, especially since corners demanded particular attention due to directional and lane changes. The result is a polished, visually cohesive game that captures the essence of Princeton's energy and challenges from a student's point-of-view. Players will appreciate the subtle humor and thematic elements, including the GPA mechanic tied to gameplay performance, with the final score calculated by multiplying the distance with the final GPA. In addition, the leaderboard allows different players to compare their performance and adds a competitive twist to the game.

We recognize that not everyone's experiences at Princeton are the same and the game does not represent a judgement on the University as a whole in any way. Instead, we tried to humorize some of the key elements that many students at Princeton are exposed to: grades, construction, the honor code, coffees, and a neverending feeling of running towards an unknown goal for unknown reasons. We also recognize that our model resembles the dynamics of other widely known and acclaimed games such as *Temple Run*. However, we did not use any code, or closed source resources and apart from the general idea of a themed "endless running game". We see our game to be in line with other contributions to the same area such as popular mobile games like Minion Rush, Sonic Dash, and Subway Surfers.

[1] https://imangistudios.com/thegames/temple-run/

3. Goal

What did we try to do? The goal of our project was to create an engaging and polished endless runner game, *Princeton Run*, that captures the unique experiences and challenges of being a Princeton student. We also aimed to leverage the skills and techniques learned throughout the semester in COS426, including procedural generation, object collision detection, and graphical rendering using JavaScript and THREE.js. Our goal was not only to develop a technically impressive and visually cohesive game but also to make it enjoyable and relatable for players. We wanted to strike a balance between humor, challenge, and simplicity, ensuring the game was accessible while showcasing our technical capabilities. Finally, we hope that *Princeton Run* could serve as a creative and meaningful project to share with peers, family, and future recruiters.

Who would benefit? Princeton Run can benefit a variety of audiences, in particular:

1. Recently admitted high school seniors: Prospective students that want to get a feel for their new university would be able to see the priorities and thoughts of current Princeton students in a low-stakes, casual way.
2. Students considering enrollment in COS426: Current Princeton students interested in computer graphics would be exposed to some of the actual work that can be completed with the knowledge from this course, allowing them to make an informed decision when they consider enrolling in COS426.
3. Potential recruiters: Recruiters may want to get a sense of our technical abilities and creative problem-solving in game development, and *Princeton Run* displays many different areas of expertise within our group.
4. Ourselves: The arguably most important group is ourselves, as we benefited greatly from this experience. It showed us how to collaborate effectively as a team, apply advanced programming and graphics techniques in a real-world context, and tackle complex development challenges.

4. Previous Work

What related work have other people done? The endless-runner category has been a popular choice for games throughout the years, combining procedural generation with obstacle avoidance in many different forms. The initial creator of this category is debatable, as it depends whether 2D games are included. Arguably, the first vertical scrolling game, *Speed Race* by Taito in 1974 could be considered the first to create and implement this idea, while Chris Pirih's *SkiFree* in 1991 may be considered a true "endless runner" game[2]. While many other games, both mobile and console have built on these precursors, *Temple Run* (2011) has been our primary inspiration due to the nostalgia it brings, as we all enjoyed playing it when we were younger. However, as *Temple Run* is a very polished mobile game, we wanted to consider building within the framework of what we learned in COS426, so rather than taking inspiration from any *Temple Run* source code, we instead took inspiration from the games overall setups and features that make it so fun to play.

Previous infinite-runner or racing-style WebGL projects have provided a groundwork of techniques such as procedural track generation and object placement. We were drawn to a simple racing game by Github user Wu Kai[3], an endless driver game by RamiAwar[4], and another driving game called "Three.js City Car Driving" by mauriciopoppe[5] as well as past experience with Temple Run, Subway Surfers, and other games that focus on running from an enemy and avoiding obstacles with the goal of surviving as long as possible.

When do previous approaches fail/succeed? While modern mobile endless runner games can be very polished and have solid themes, it appears that their THREE.js counterparts are lacking in this area and aim to show basic working mechanics. Our project builds on these concepts by adding a variety of special features that allow it to stand apart from other THREE.js endless runner implementations: a Princeton University-themed layer, rich variety of obstacles, day-night cycles, event-driven sounds, and a scoring system tied to both distance traveled and performance metrics (GPA). Other THREE.js implementations also typically include only a

---

[2] https://en.wikipedia.org/wiki/Endless_runner
[3] https://github.com/noiron/race-game-threejs?tab=readme-ov-file
[4] https://ramiawar.itch.io/nightdrive
[5] https://github.com/mauriciopoppe/Three.js-City?tab=readme-ov-file

straight road, but our implementation procedurally generates roads with turns, which the player can turn onto using the left and right turn functionalities while in the center of the turning square.

5. Methodology

To execute our approach for Princeton Run, several key components had to be implemented. For each piece, we considered multiple possibilities and selected the most appropriate solutions based on performance, ease of implementation, and the specific goals of the game.

1. *Procedural Road Generation*
    a. Implementation Options: We could have manually crafted a fixed set of road segments with predetermined turns, obstacles, and path layouts, but this would not have provided the infinite, dynamically generated experience we aimed for. Another option was to use a procedural generation algorithm that could create randomized road segments and obstacles based on certain rules (e.g., alternating turns and straight paths).
    b. Advantages/Disadvantages: Manual design would have limited replayability and would not have matched the fluidity and infinite gameplay of the endless runner style, despite being simpler to implement. Procedural generation provides a dynamic experience, but requires more logic to handle the random creation of obstacles, turns, and the deletion of outdated segments.
    c. Implementation: We chose to implement the ProceduralRoad class with dynamic generation. It spawns new road segments ahead of the player and removes old segments that the player has already moved past, ensuring continuous gameplay without compromising performance. This implementation provided the desired infinite runner experience and replayability.
2. *Player Movement and Controls*
    a. Implementation Options: The player could be controlled using either mouse movement or keyboard inputs. Given the genre of the game, keyboard input (W/A/S/D or arrow keys) was the more natural choice.
    b. Advantages/Disadvantages: Keyboard controls are standard for most endless runner games and provide more precise movement, permitting more features like jumping and enabling power-ups. Mouse control could have been intuitive but would likely require more complicated tracking and interaction logic.
    c. Implementation: We implemented the Student class to handle movement via keyboard controls. The player can switch lanes, jump, or trigger a sprint boost using coffee pickups. This choice allowed for simple yet effective gameplay mechanics aligned with the genre. Note: A/arrowleft for left lane shifts and turns, D/arrowdown for right lane shifts and turns, W/arrowup for jump, and P for powerrun (once enough coffees have been collected).
3. Obstacles and Collectibles
    a. Implementation Options: We could have manually defined a set of obstacles with fixed positions for each road segment, but this would reduce the variety and challenge. Another approach was to use probabilistic spawning, randomly placing obstacles and collectibles based on predefined probabilities, ensuring unpredictability in gameplay, with the issue that sometimes these objects overlap.
    b. Advantages/Disadvantages: Fixed positions would provide less challenge and variety, making the game feel repetitive. Probabilistic spawning increases the complexity of managing collisions and balance, but it offers the unpredictability and excitement typical of infinite runner games.
    c. Implementation: We chose to implement the RoadChunk class with probabilistic obstacles and collectible spawning. This approach created varied gameplay experiences, with different combinations of obstacles and items in each run. We found that objects do not frequently overlap due to the large number of spaces available for them to spawn, but we could disallow object overlap in future versions of the game.
4. Collision Detection

    a. Implementation Options: We decided between bounding box checks and a more complex approach that used intersection on a bounding sphere/ellipsoid or pixel by pixel basis, which might produce more realistic results.
    b. Advantages/Disadvantages: Pixel-based detection could be inefficient for a 3D environment, especially with moving objects. Bounding box collision detection is simpler to implement and efficient, as it checks for overlaps between rectangular regions around objects.
    c. Implementation: We implemented bounding box collision detection for each object in the game. This method works well for our needs, efficiently detecting collisions between the player and obstacles, and triggering the appropriate game state changes (e.g., losing lives, collecting items). However, we note that sometimes the player "hits" an obstacle when he is in close proximity. We decided to trade simplicity for realism.
5. Scoring
    a. Implementation Options: The scoring system could have been based solely on distance or based on the collection of items as well. GPA could be tied to distance traveled, or we could tie it more directly to the number of grades collected. There are an infinite number of possible options here.
    b. Advantages/Disadvantages: Tying the GPA to distance gives players an incentive to keep running, while encouraging players to explore the road and collect items, making the gameplay more interactive. A purely distance-based system would have been simpler but less engaging.
    c. Implementation: We implemented the GPA system based on the grades collected and tied it to the final score, which is a product of both distance and GPA. This system encourages players to collect grades while avoiding obstacles. It is also relatable for many of the intended users. In addition, collecting a "generative AI box" sets a player's GPA automatically to 4.0 (at a small risk of immediate death! See more in the section below)
6. Lives System
    a. Implementation Options: As with any game, there exists a long list of implementation possibilities along various dimensions. We had the option to vary the number of lives, how they get regenerated (if at all), what items regenerate them etc.
    b. Advantages/Disadvantages: Similar to the scoring section, we believe, there exists a reverse relationship between complexity of the system and user experience. In addition, more lives might extend a single run but might make the user focus less on obstacles. Fewer lives might have the reverse effect where a user is "too afraid" to collect coffees, grades, etc. out of fear of hitting an obstacle and dying. Similarly, we expected regenerating lives too quickly/slowly to have similar effects.
    c. Implementation: We decided to give a player three lives per game which is a common number. The lives can be regenerated at a certain risk (picking up a generative AI box) which complicates decision-making for the player (should I risk it, if I still have 2 lives?). Hence, the genAI box has two functions: regeneration of lives and setting the GPA to 4.0. We believe this to be a funny comment on the current feeling of a lot of students around using these emerging technologies: Using ChatGPT might be really helpful, but I might shoot myself in the foot.
7. Visual Enhancements and Thematic Elements
    a. Implementation Options: We could have kept the game visually simple with basic textures and static backgrounds, but that would have lacked the immersive and thematic experience we wanted. A more complex approach could have involved dynamic changes in lighting, and more themed assets to enhance the Princeton atmosphere (i.e buildings alongside the road).
    b. Advantages/Disadvantages: Basic visuals would have been easier to implement, but they wouldn't have captured the essence of Princeton or made the game feel as engaging. Using dynamic skyboxes and textures adds complexity but greatly improves the visual appeal and immersion of the game.

c. Implementation: We chose to implement dynamic changes in the 2D background (day/night cycles) as well as clouds appearing in the sky above the runner and thematic elements (Princeton-related obstacles and collectibles). These visual enhancements strengthen the game's connection to Princeton and improve the overall player experience. Additionally, sounds are added to help the player understand 'good' and 'bad' objects to a degree if the start screen instructions are unclear to them. However, we did not have time to add more objects alongside the road, such as the ground of the campus, as the 2D ground implementation broke the illusion of moving through campus as the runner and camera moved forward, while the 2D background remained stationary.

8. Database
    a. Implementation: We had the option to implement a local leaderboard, global leaderboard, or no leaderboard . A leaderboard can keep track of top scores per player, top scores overall, top GPAs, etc.
    b. Advantages/Disadvantages: Similar to other sections, we faced a choice between added complexity and user experience. Keeping track of more data (especially for individual best scores, etc.) was more difficult compared to just keeping track of overall best scores. However, implementing a global leaderboard also adds a competitive objective to the game that may convince players to continue playing to try to beat the top score.
    c. Implementation: Since we have very little (to not say zero) experience with databases, we settled on doing the simplest possible thing: keeping track of the top scores for all players. If a player beats one of the top three scores, they get a medal at the end and the leaderboard gets updated to reflect the new score. This was a last minute addition so we are not confident in it functioning consistently once we scale to many users (during our testing, it worked for 3 simultaneous users).

9. Results

How did we measure success? We measured the success of our game by the extent of a player performing unintended actions or causing unintended effects. A successful game in our case is one where the user is able to easily act as intended (ie. running and jumping to avoid obstacles, turning to stay on the path) without prior experience playing the game.

a. What experiments did we execute? While developing *Princeton Run*, we tried to force as many corner cases as we could and account for them. For example, while working on the turns, we found and fixed issues regarding being able to turn multiple times within a single corner, continuing to run once off the edge of the map, and jumping onto different roads. In terms of players with no experience playing the game, we had several others try the game and took their input into account, such as having less obstacle generation initially while new players become acclimated to the game.
b. What do my results indicate? We have created a mostly bug-free game that is playable and fun for all skill levels. Beginners will find detailed instructions on the start screen and have the ability to get acclimated to the game initially, while experts who have already played the game several times will not get bored or begin to memorize the game due to the procedural road generation and random object spawning throughout each game.

10. Ethical concerns.

In this section we want to comment on potential ethical concerns of our game. We identify two potential aspects that could be objectionable to someone. The first ethical concern is on Representation and Inclusivity (ACM Ethical Principles 1.4). The game includes stereotypical representations of what a "Princeton experience" looks like. However, it definitely fails to reflect the diversity of real-world students. For instance, the game only shows a narrow depiction of students (e.g., by using a single gender or ethnicity for the player character). The ACM Code of Ethics (1.4) emphasizes the importance of "foster[ing] fair participation of all people" and avoiding discrimination based on factors like gender, ethnicity, or race. To mitigate this, we think that the next version of the game should contain three key additions. First, it should offer customizable

characters that allow players to choose their appearance (more inclusivity). Second it should allow users to opt-out of certain objects (i.e. honor code). Third and last, we would like to implement a feedback function for players to express their grievances and tell us possible expansions of the game to make it more inclusive. By addressing these aspects, we would ensure the game respects diversity and fosters a fair and inclusive experience for all players.

The second ethical objection someone could have concerns respecting intellectual property and creative works (ACM Ethical Principles 1.5). As our game incorporates assets like textures, models, and music, it's important to ensure that all resources used in development are properly accredited and licensed. Using assets without permission, or failing to credit original creators, could violate intellectual property rights. The ACM Code of Ethics (1.5) emphasizes that computing professionals must "respect the work required to produce new ideas, inventions, creative works, and computing artifacts." This principle ensures that creators are appropriately recognized and compensated for their efforts. We did our best to document which assets we obtained code from other developers even in places where we were not required to do so. By respecting these guidelines, we acknowledge the effort and creativity of others, fostering a professional and ethical development process.

11. Conclusion

How effectively did we attain our goal? We can confidently say that we did way more than we expected (in part because the project was very fun). We are extremely happy with how *Princeton Run* turned out, especially considering how much more we accomplished than we initially anticipated. Not only did we create a fun and engaging game, but we also implemented many advanced features such as procedural road generation, dynamic player movements, and Princeton-themed obstacles. The game captures the spirit of student life at Princeton in a humorous and relatable way, while also demonstrating our technical abilities.

What would the next steps be? Looking ahead, and potentially as future work (or if we had had more time), there are several things to pursue. First, to make the game more immersive, we would like to add buildings along the road to enhance the environment and create a more realistic cityscape. We would also address the issue of obstacle placement to ensure that they do not intersect, improving gameplay flow. Additionally, making the code cleaner and more modular would help in future development and maintenance. Enhancing the starting and end scenes would provide a more polished user experience, and we could introduce new power-ups to make the gameplay even more dynamic and enjoyable. All in all there is a lot that can be done. Nonetheless we are very happy with the progress we were able to make in one and a half weeks.

What are the issues we need to revisit? One major issue is the overlap between objects in the game, which can occasionally disrupt the smooth gameplay experience. We need to fine-tune object collisions and improve the placement logic to ensure a more polished, bug-free interaction. Addressing these and other small refinements would further enhance the game's quality and user experience. There are also occasional weird bugs like odd turns (they happen extremely rarely and we are not quite sure if they are a bug or simply due to us not hitting a turn at the appropriate time).

12. Contributions

Max (i had a lot of fun with this lol):

- implementation of the road system (generation, sidewalks, ups and downs, etc.)
- implementation of turns and jumps
- implementation of sprinting + coffee system
- implementation of database + leaderboard system
- implementation of obstacles + scoring system
- implementation of runner behavior (adjusting runner dynamics to speed/jumps. Note that the glb file contained the animations)

Alyssa

- creation & implementation of backgrounds for day/night cycles

- implementation of initial start screen
- creation & implementation of moving clouds
- creation & implementation of sounds and music
- modification & creation of meshes as needed (see References)

Ava

- implementation of obstacle collision detection (via bounding box method)
- implementation of ending screen
- implementation of lives system

13. References

Overall code structure:

- We want to pay particular tribute to: https://harveyw24.github.io/Glider/ as it helped us understand the initial codebase better and provided some implementation ideas around how to structure the code and potential extensions. We did not use any code but their implementations, nonetheless, helped kickstart our project.

Runner GLB file:

- CREDITS: Animated Human by Quaternius (https://poly.pizza/m/c3Ibh9I3udk); colors modified via Blender 4.3

Coffee GLB file:

- CREDITS: Coffee cup by Poly by Google [CC-BY] via Poly Pizza (https://poly.pizza/m/fIuM_PW5prV)

Oak and Bush:

- Used procedural generator published by: https://github.com/dgreenheck/ez-tree?tab=readme-ov-file

Bike:

- CREDITS: Bicycle by Poly by Google [CC-BY] via Poly Pizza (https://poly.pizza/m/19VoUuA2pcN)

Clouds and Backgrounds:

- Created via Procreate iOS app

Grades A+ through F

- Created via Blender 4.3

GPT:

- Picture for Claude: https://brandfetch.com/claude.ai
- Picture for ChatGPT: https://media.licdn.com/dms/image/v2/D5612AQGNFfMWKQZSSg/article-cover_image-shrink_720_1280/article-cover_image-shrink_720_1280/0/1688612205621?e=1739404800&v=beta&t=vhy8VMddHkExjJCvSC81ozckiK07Gme003ZsyibfPEc

Road:

- https://stock.adobe.com/images/granite-cobblestoned-pavement-background-stone-pavement-texture-abstract-background-of-cobblestone-pavement-close-up-seamless-texture/403115894

Sidewalk:

- https://www.google.com/imgres?q=sidewalk%20texture%20cobblestone%20dark%20brown&imgurl=http%3A%2F%2Fwww.sketchuptextureclub.com%2Fpublic%2Ftexture%2F0057-street-paving-cobblestone-texture-seamless.jpg&imgrefurl=https%3A%2F%2Fwww.sketchuptextureclub.com%2Ftextures%2Farchitecture%2Froads%2Fpaving-streets%2Fcobblestone%2Fstreet-paving-cobblestone-texture-seamless-07389&docid=On-TF9Uinv7jtM&tbnid=VdW8x694cKQ3zM&vet=12ahUKEwir19i

FkKWKAxWHF1kFHS0qKsQQM3oECGMQAA..i&w=1200&h=989&hcb=2&ved=2ahUKEwir19iFkKWKAxWHF1kFHS0qKsQQM3oECGMQAA

Wall:

- CREDITS: Wall by Quaternius (https://poly.pizza/m/CkF171SeTV)

Tiger:

- CREDITS: Tiger by Poly by Google [CC-BY] via Poly Pizza (https://poly.pizza/m/5A3w06FXUup)

Cone:

- CREDITS: Traffic Cone by Quaternius (https://poly.pizza/m/aDIrUbMbW3)

Rights, Rules, Responsibilities Book:

- CREDITS: Book by Quaternius (https://poly.pizza/m/LC0w7VI75u); modified to add title via Blender 4.3

Construction Sign:

- CREDITS: Construction sign by koen caspers [CC-BY] via Poly Pizza (https://poly.pizza/m/4v9QhryzBwT)

Sound effects:

- Created via jsfxr (sfxr.me)

Background & powerrun music:

- Created via Chrome Music Lab (https://musiclab.chromeexperiments.com/Song-Maker)