William Trojniak, Michael Huang
COS426 Final Project

Pogo Parkour

Abstract:

In this project, we created a 2D platformer where the player traverses a number of levels and avoids spikes in order to reach the end. A core mechanic of our game is the "pogo" mechanic, where the player can jump in midair by pressing an action key over a spike. Having implemented our own physics and collision detection, we aim to provide a simple yet fun experience for any who chose to play our game.

Introduction:

2D platformers are a very popular subcategory of games where the player controls a character that moves left to right on the screen, usually from a side scrolling perspective. The goal of this project was to design our own 2D platformer and implement the game mechanics from scratch. We designed and created each level ourselves, and we also wrote code to implement the physics of the character's movement on the screen as well as detect collisions with both solid blocks and spikes. A core mechanic of our game was the "pogo" mechanic. A collision with a spike would normally kill the character and send it back to the start of the level. However, if the player were to press an attack key right above a spike, a sword slash animation would appear and hit the spike, propelling the player upwards in the air. This mechanic gave us many new avenues of creativity for the level design, as spikes represent both danger and an opportunity to progress the level. The project would benefit anyone who enjoys 2D platformers and wants to both relax and challenge themselves with a simple game.

Given the popularity of 2D platformers, there have been a vast number of games built in this style. In particular, the pogo mechanic was inspired by the popular 2D platformer Hollow Knight, where the character is capable of bouncing on top of spikes using a nail slash. Our initial idea of creating a 2D platformer was inspired by the Cave Climber submission in the COS426 Hall of Fame submissions from 2021. There are many different ways to approach creating a 2D platformer. One approach includes keeping the character constant on the screen and moving the background behind it, thus granting the illusion of movement. Another approach is allowing the character to move around on the screen, but allowing the background to move with the character once it reaches a certain bound near the edges of the screen. In terms of frameworks, 2D platformers can be made with a wide variety of frameworks including Three.js, Unity, and much more. Furthermore, though some approaches may be more suited depending on the style and mechanics of the game, the simplicity of 2D platformers allows most approaches to be successful.

For our approach, the first decision to make was what framework to develop the application in. Though many COS426 games used Three.js, which was also a viable option, we decided to go with Pixi.js, a framework optimized for 2D graphics and rendering. For the 2D platforming, we went with the popular method of creating a tilemap for each level. By manually creating a large array of ids, we can create each level by looping over the array and rendering the associated texture with each id. This approach works well in our game because we wanted the level to be strictly constrained to the screen, so the rigid implementation of the tilemap would help us create the level design.

Methodology:

1. Scenes and Levels:

The first piece that we had to figure out was how to implement the scenes and the levels. First of all, we needed to get the assets and art of the game. We used assets from the website opengameart.org, which freely allows game developers to use art for their own projects in adherence to the license terms. We used basic models and animations for the character, the blocks, the spikes, and more. Once we loaded the art into our application, we created an atlas that would store the specific pixels for each block along with an id to identify the blocks. (GET URL OF ART ASSETS) For the level design itself, as described above, we went with a tilemap approach.

```
tiles:
  [
    11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12,
    10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    10, 0, 4, 5, 6, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 8, 0, 0, 0, 0, 9,
    2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  ],
```

Figure 1: TileMap for First Scene

As seen in Figure 1, the tilemap is just an array of numbers, with each number corresponding to a specific texture. The tilemap also contains the number of rows and columns in the level. When we want to render the scene, we loop through the array, using the row and column number in order to find the place on the screen to place the tile and using the id to find the correct texture at that tile. The advantage of the tilemap approach is that we have a highly customizable level layout, and since the tiles are all squares, we can ensure that every part of the level is covered. Furthermore, it's very simple to change the look of a specific part of the level because all we need to do is change the corresponding id at that point. A disadvantage of this method is that with our particular implementation we are unable to extend the scene off the screen, limiting the amount of content we can put in each level at a time. Had we implemented the tile map with content off the screen, we would've had more space for each level but it would've required more complex work with camera movement and player tracking. For simplicity, we chose our implementation to have all the content on the screen at once.

2. Physics & Collisions:

A major stepping stone and challenge in creating our game was handling physics and collisions. Given the experiences from the class projects working with simple physics and collision detection systems and their simpler nature in 2D settings, we decided that we wanted to try implementing our own engine, rather than a prebuilt library. Our system uses an ECS-like

approach where any object that might collide with other objects has a collision box component. For instance, from the client's perspective, getting an object to move and collide with other objects in the scene is as simple as adding a physics body component to the object that should move and adjusting the speed and forces acting on it each frame. Behind the scenes, the creation of these objects are registered with the physics and collisions system which performs checks between all of the collision boxes and exposes callbacks to the client to handle them. There were many challenging steps in creating this system, the most challenging of which was preventing bounding boxes from clipping into each other, particularly when the player was actively walking into a wall, for instance. We solved this issue by first applying our physics updates before then checking for collisions and adjusting any physics bodies to just outside of the clipping the bounding box. The major drawbacks of this system are its limited functionality and performance, especially given our naive approach of checking collisions between every box in a scene. However, we architectured our scenes such that each one is quite simple and has its own physics engine. That way, the number of possible collisions is kept relatively low, and the engine works more than well enough for our needs. Moreover, it was an incredibly rewarding experience to build it and see our player move and collide (eventually properly) with it.

3. The "Pogo" Mechanic:

As previously mentioned, the key mechanic that we wanted to implement in our game was a "pogo" mechanic. Essentially, the mechanic involves the player timing an input such that their player is vertically boosted when done correctly, with the caveat that the player can only do so above "spike" tiles which reset the players progress when collided with. This mechanic allowed for many different game design decisions. In particular, it provided an avenue for skill expression in the form of proper timing, not only with the input but also of successive "pogos" in a row given the mechanic's cooldown time, and also provided players with choice when faced with multiple routes in a level. Implementing the feature built on all of the previous systems we created. The slash was given its own collision box and the engine was used to check for collisions between it and any spike collision boxes. The players' callbacks were then used to check for this collision and if the player had requested a slash action, then the physics engine was used to give the player an impulse upwards. The mechanic also used small animation controller systems that we implemented on top of the animations from Pixi.js in order to achieve the player's animations for this mechanic.

Results:

We eventually created the game and successfully deployed it. The levels work as intended and both the physics and collisions work well. We measured the success of the game if the player is able to make it to the end without running into major bugs. After testing the game ourselves, we found that we both were able to make it to the end with all the features working as intended. Furthermore, we both enjoyed playing through the game and testing the mechanics, and we counted this as a big success since our primary goal was to make a fun game. The experiments we executed were mostly just playthroughs of the game. We tested the collisions in extensive edge cases, such as collisions with walls, the underside of blocks, the side of spikes, and much more. We also tested to see if the pogo mechanic worked as intended, testing the frequency of the slashes and seeing if the slash collision with spikes consistently propelled the character up. We ran into many bugs while doing our experiments and we were able to fix them and polish the

game. Our results indicated that extensive testing and evaluation is the only way to find all the bugs in a game, and even though there undoubtedly some we weren't able to catch, we overall were able to eliminate the bugs to a point where it wouldn't significantly take away from the experience of the game.

Ethical Concerns:

The first aspect of the project one might have objectionable concerns over is the degree of originality. We created the level design, physics, collisions, and mechanics for the game ourselves, and we came up with the idea for the project on our own. However, we were still inspired by many 2D platformers, most notably Hollow Knight for the pogo mechanic. Part of the ACM guidelines is to respect the work required to produce new ideas and inventions. Therefore, someone might look at our project and deem it too similar to other 2D platformers, thus raising an ethical objection. One very easy method to mitigate this objection is to simply give credit to games we were inspired from. By doing so, we don't rob others of their originality. Furthermore, one of the ACM guidelines is to credit the creators of ideas and respect copyrights.

Another aspect of the project that could cause an objectionable concern is the difficulty of the game. Some of the jumps we put into the game were purposefully difficult in order to pose a challenge to the player. However, some players might become too focused on completing these jumps and eventually lose the original purpose of simply having fun. In these cases, such challenges might even become addictive to some degree. Therefore, this would violate the ACM principle of using skills to the benefit of all people. Something we can do to mitigate this is creating other paths and routes so people won't get too frustrated with the game and will still be able to have fun even if they can't complete the most difficult portions.

Discussion:

The approach we took was promising. Pixi.js is a very well known and documented framework for 2D games, and the tilemap implementation also has a high rate of success in other games. Throughout the course of the project, we were able to find bug discussions and documentation online to help us progress in the development. When reflecting on the overall process, we realized that most of our time was spent on developing the physics and collisions. If we had used a physics engine online, it would've allowed us to chart out more complex scenes and add more mechanics into the game. Therefore, though our approach allowed us to learn more about 2D graphics and game development, if we had approached the project with an existing physics engine we probably would've been able to create a much more intricate game. Overall, we still learned a lot about how to implement physics and collisions, so we believe that our approach was ideal.

Our game still has a lot of room to grow. If we had more time, we had ideas for extending the number of scenes and adding more mechanics for the player to explore. As an example, we considered adding moving platforms and moving spikes to make the scene more dynamic and further emphasize the importance of timing and careful movement. We also could have added enemies to the game capable of moving around and damaging the character.

By completing this project, we learned a lot about 2D graphics, TypeScript, rendering, Pixi.js, game development, physics engines, and so much more. Besides the art and audio, we implemented everything from scratch. Therefore, through careful trial and error, we learned the best methods and practices for keeping our code modular and efficient. We learned how to render everything on the screen through Pixi.js and how to display the components relative to each other and the player. By creating the physics and collisions ourselves, we explored the best ways to efficiently keep track of the character's interactions with the level and how to update the character's movement. When we conducted our own experiments and playthroughs, we were able to implement some features and fix some problems we didn't think about during development, allowing us to learn how to effectively test and debug our application.

Conclusion:

Our final project is a simple 2D platformer that we completed from scratch and is focused on being fun and interactive for the player. We both enjoy playing the game and learned a lot about graphics and game development, so we believe that the project was a success. Though there were some roadblocks and issues, we were able to effectively finish the project and produce a working version. Overall, we had a lot of fun with this project and we are proud of the end result. If we had more time, the next steps would be to add more mechanics and levels to the game without losing out on the simplicity of the game. There are still some minor bugs in the game and some improvements we could make, but nothing major enough to cause issue with someone playing through it.

Will developed the physics, collision, scene switching, and animation mechanics. Michael implemented the tile maps and designed the levels. Both of us worked on minor miscellaneous features and the writeup.

References

- https://opengameart.org/
- https://opengameart.org/content/75-cc0-breaking-falling-hit-sfx (Spike SFX by rubberduck)
- https://opengameart.org/content/platformer-jumping-sounds (Jumping SFX by dklon)
- https://opengameart.org/content/a-platformer-in-the-forest (Character and Various Tile models by Buch)
- https://opengameart.org/content/castle-platformer (Spike Tile by Jetrel)